UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

UNIDAD DE POST-GRADO

Un algoritmo GRASP con simulación dinámica para resolver el problema de cortes irregulares

TESIS

para optar el Grado de Magíster en Ingeniería de Sistemas

AUTOR

Fernando Arturo Torres Sánchez

Lima - Perú

2007

"*A la memoria de Fernando Torres Santa Cruz*"

Agradecimientos

Este proyecto es el fruto de una constante dedicación y esmero durante mas de tres años y medio de trabajo. Asimismo, punto de partida de un camino que aún falta por recorrer.

En primer lugar deseo agradecer a mis asesores, el Dr. David Mauricio Sánchez y el Dr. Luis Rivera Escriba, por su amabilidad, interés, constante apoyo y orientación; aprovecho la oportunidad para expresarles mi entera gratitud por su amistad en todo momento.

En segundo lugar a mis padres a quienes debo mi educación y formación; a mi madre, por el esfuerzo y sacrificio, y a mi padre quien en vida me brindó el aliento para seguir adelante.

A mi novia Alida Anicama por su apoyo para poder culminar este trabajo; a mi hermana Fanny y Karin Salvatierra por la corrección gramatical. A algunas pocas amistades que confiaron desde el inicio en la realización de este proyecto a Milagros, Billy y Abigail por su amistad y sincero cariño.

Resumen

Los problemas de corte de piezas se presentan en diversos sectores productivos y han sido ampliamente estudiados en la literatura. En algunos casos, son problemas simples de especificar; pero en todos los casos son extremadamente difíciles de resolver; aqui se considera un caso de esa familia de problemas denominado el problema de corte de piezas irregulares.

El objetivo principal de este trabajo es implementar un algoritmo que permita resolver el problema de corte de piezas irregulares el cual consiste básicamente en minimizar el número de superficies que alojen a una determinada cantidad de piezas irregulares en demanda, posicionar las piezas en las respectivas superficies, permitiendo minimizar la pérdida de material.

La solución a este problema se entrega a través de un programa computacional basado en la metaheurística GRASP y la Simulación Dinámica, donde el algoritmo implementado selecciona las piezas irregulares a posicionar en cada superficie, para luego aplicarles parametros físicos necesarios para la simulación dinamica. Una vez que las piezas se compacten dentro de cada contenedor se tiene la configuracion final, donde se analiza el desperdicio resultante.

Se midió el rendimiento del algoritmo basándose en un conjunto de instancias de prueba; los experimentos numéricos sobre 10 instancias numéricas muestran un bajo costo computacional y un grado de eficiencia respecto a la calidad de la solución en 100 %.

Palabras Claves: Problema de Corte de Piezas Irregulares, Simulación Dinamica, GRASP.

Abstract

The problems of cut of pieces appear in diverse productive sectors and widely have been studied in Literature. In some cases, they are problems simple to specify; but in all the cases they are extremely difficult to solve. In this work a case of that family of problems is considered: Irregular Cutting Stock Problem.

The primary target of this work is to implement an algorithm that allows to solve the problem of cut of irregular pieces, which consists of diminishing the number of surfaces that lodge to a certain amount of irregular pieces in demand, to position the pieces in the respective surfaces, being allowed to diminish the lost one of material.

The solution to this problem is given through a computer program based on GRASP metaheuristic and the Dynamic Simulation, where the implemented algorithm selects the irregular pieces to position in each surface, soon to apply necessary physical parameters to them for the dynamic simulation. Once the pieces are compacted within each container has the final configuration, where the resulting waste is analyzed.

The performance of the algorithm was moderate being based on a set of test instances; the numerical experiments on 10 numerical instances show to a low computational cost and a degree of efficiency with respect to the quality of the solution in 100 %.

Keywords: Irregular Cutting stock problem, Dinamic Simulation, GRASP.

Índice general

Índice de figuras

Capítulo 1

Introducción

Desde hace años atrás, con el surgimiento de la Inteligencia Artificial se ha intentado resolver problemas de las más diversas índoles; problemas de características muy elementales (fáciles de resolver), hasta avanzadas aplicaciones que juegan un papel muy importante en el sector industrial y tienen la característica de ser duros.

Las grandes industrias han estado en continua competencia tratando de preservar la calidad de sus productos. Debido a ello, están buscando siempre tener la mejor mano de obra calificada, así como una excelente tecnología de punta que les permita reducir al máximo la pérdida de materia prima en sus procesos.

El problema de cortes es un proceso clave en muchas industrias de manufactura; por ello se sabe que si una industria es capaz de minimizar la cantidad de pérdida de material que produce, existiría un ahorro cuantificable en sus costos de producción. Asímismo, la industria podría ser capaz de reducir su capacidad de almacenaje generándose un mayor ahorro en costos.

Es así que en muchos procesos industriales, se da el problema de cortes (cutting stock problem) (tambien llamado Trim-Loss por algunos autores) Dyckhoff [Dy90], Gilmore-Gomory [GG65]. A lo largo de los años, este problema ha incrementando su importancia y atención, ya que al resolverlo adecuadamente se minimizan costos, pérdida de material, niveles de inventario final y tiempo de máquina. En nuestro caso, el problema a tratar se enfocará a industrias en las cuales las demandas de las piezas posean forma irregular, como son la industria de calzado, ropa, cartón, fabricación de máquinas, etc.

Es de gran importancia para este sector industrial el ahorro del material que se utiliza para conseguir las piezas que se requieren. Hasta ahora la cantidad de pérdida de material en los procesos de corte depende de la habilidad del experto humano que, confiando en sus experiencias, determina la mejor forma de cortar el material entero que comúnmente suele llamarse "planchón", ocurriendo siempre desperdicios. Para dar solución a este problema es preciso desarrollar una metodología, la cual permita ahorrar el desperdicio de material, trayendo consigo mejora de procesos y, por ende, una disminución en el costo de producción.

El tipo de material a considerar es una superficie geométrica tipo planchón, el cual es una lámina de área A, que en muchos casos dependiendo del tipo de material es rectangular de lados L y H, existiendo también planchones no rectangulares como es el caso del cuero.

La forma de los requerimientos(piezas de corte) no siempre es regular, esto quiere decir que existen requerimientos de formas rectangulares o poligonales; estos pueden ser de geometría compuesta de pedazos de curvas y rectas, en general determinados por molduras pre-definidas, tal es el caso de las piezas para zapatos que es casi imposible encontrar una pieza rectangular en sus componentes.

El presente trabajo de tesis es perfectamente aplicable en la industria de nuestro país, en los sectores industriales en los cuales se destaquen los rubros textiles, madereros, etc; ayudando a la mejora de su productividad, mejorando su eficiencia y disminuyendo el tiempo y los factores de error que puedan existir.

1.1. Motivación

Es de alto interés para las industrias de producción masiva de material el poseer mecanismos de corte adecuados para conseguir mejoras en su productividad, lo cual resultaría en un ahorro de material y en una considerable disminución en sus costos de producción. Hasta ahora muchas industrias vienen realizando esquemas de corte de manera manual lo que les dificulta un ahorro de material al realizar este proceso de manera empírica.

Como bien es sabido, en los procesos de cortes existe una pérdida considerable de materia prima. Con el presente trabajo de tesis se busca la reducción del desperdi-

cio ocasionado en los procesos de cortes irregulares en las industrias, lo que traería consigo una reducción significativa de los costos de producción y el aumento de la competitividad.

Por la naturaleza irregular de las piezas, hasta la actualidad no existe alguna técnica que resuelva el problema de manera eficiente. La alternativa propuesta esta basada en la técnica GRASP con simulación dinámica; donde la técnica GRASP minimizará el número de superficies a utilizar para una determinada cantidad de piezas en demanda y seleccionará las piezas en cada superficie; la simulación dinámica, ayudará considerablemente para la compactación de las piezas en el recipiente.

Los usuarios potenciales directos e indirectos que se beneficiarían con el resultado del presente trabajo serían:

- Las industrias que usan como materia prima las planchas de cuero: fabricantes de zapatos, bolsas, tapetes, casacas, entre otros.

- Los fabricantes de artículos de metal, tipo latones sin una orientación de brillo o alguna señal física o de formación molecular, pueden ser los usuarios de este producto.

1.2. Propuesta de Investigación

El presente trabajo de tesis tiene como objetivo formular un método que reduzca el desperdicio ocasionado en la industria de cueros por los procesos de cortes irregulares en dos dimensiones; así como desarrollar un software altamente eficiente para la Gestión de Cortes Irregulares que permita minimizar el desperdicio ocasionado por la gestión de cortes en dos dimensiones.

Para esto se desenvolverán métodos metaheurísticos de inteligencia artificial y algoritmos de compactación de objetos irregulares por leyes de la física. Se conoce que el problema que se está tratando es de naturaleza NP-Difícil (difícil de resolver por un algoritmo exacto), ya que computacionalmente el encontrar su solución tomaría demasiado tiempo.

Se pretende realizar un método GRASP con simulación dinámica el cual se desarrollará en dos etapas para realizar la distribución de las piezas en las superficies contenedoras:

1) Selección de las piezas que se alojaran en las superficies contenedoras con respecto a los requerimientos que se demanden.

2) Se realizará iterativamente el proceso de compactación hasta llegar a una adecuada configuración final [Ri01], consiguiéndose en este momento una solución de calidad que resolverá el problema. Para conseguir una compactación aceptable se tendrá que buscar una forma en la cual se muevan las piezas sin alterar el estado local de sus posiciónes. Asímismo se realizará un análisis de las areas finales luego de la compactacion usando conceptos de geometría computacional avanzada.

Se hará uso de la animación gráfica para poder visualizar todo el proceso de la solución del problema, mostrándose visualmente desde la configuración inicial de las piezas en una superficie hasta su compactación.

1.3. Contribuciones

Se espera producir un programa informático eficiente y robusto que permita decidir la forma como se debe cortar planchas en pedazos irregulares, minimizando el porcentaje de desperdicios.

Se considerarán piezas irregulares con bordes redondeados, enfocándonos más que todo al sector industrial que requiere de corte de piezas irregulares.

1.4. Organización del Trabajo

El presente trabajo esta estructurado en 10 capítulos. El Capítulo 2 es dedicado a la revisión en la literatura del problema de cortes irregulares, mostrándose las técnicas que han sido estudiadas a lo largo de los años para resolver este problema; así como se define el método de solución.

El Capítulo 3 es dedicado a la definición de la arquitectura modular que seguirá el

método propuesto para resolver el problema. En este capítulo se define el funcionamiento de cada módulo y la comunicación que se presentara entre éstos.

En el Capítulo 4 es abordado con detalle la técnica GRASP con simulación dinámica a implementar. En este capítulo se muestra el algoritmo GRASP, su funcionamiento, se presentan detalladamente sus componentes, las partes principales de su diseño, funciones y procedimientos.

El Capítulo 5 muestra en detalle el método utilizado para realizar la configuración inicial de las piezas en demanda. Son estudiados en detalle los métodos de posicionamiento que existen en la literatura y es propuesto el método Bottom Left para conseguir este propósito.

En el Capítulo 6 es tratado la generación de movimiento de las piezas en la superficie contenedora por medio de la simulación dinamica. Es aquí donde se definen los parámetros físicos de cada objeto, deteccion de interferencias durante el movimiento, la dinámica de colisiones basada en impulsos hasta llegar al estado de compactación.

En el Capítulo 7 se realiza el análisis de compactacion de las piezas. En este capítulo se analiza el estado final de las piezas dentro de la superficie, calculando el cerco convexo que generan para luego determinar el area de utilización.

El Capítulo 8 muestra detalladamente al descripción del software y hardware utilizado para el desarrollo del programa así como una descripción de sus componentes principales.

En el Capítulo 9 se presentan las instancias de prueba por las que se ha hecho pasar el algoritmo y los resultados numéricos que han sido consecuencia de las pruebas.

En el Capítulo 10 son hechas las respectivas conclusiones generales y particulares referente a los módulos del sistema así como tambien son detallados los trabajos futuros.

Capítulo 2

El Problema de Cortes Irregulares en dos Dimensiones

Durante muchos años, las grandes industrias han estado en continua competencia tratando de preservar la calidad de sus productos. Debido a ello, están buscando siempre tener la mejor mano de obra calificada, así como una excelente tecnología de punta que les permita reducir al máximo la pérdida de su materia prima.

El problema de cortes es un proceso clave en muchas industrias de manufactura; por ello, se sabe que si una industria es capaz de minimizar la cantidad de pérdida de materia prima, existiría un ahorro cuantificable en sus costos de producción. Al minimizar el desperdicio de materia prima, la industria incrementa sus lucros por los productos manufacturados. También, una forma eficiente de cortar la materia prima, permite optimizar el tiempo de uso de las máquinas, de los técnicos y operarios comprometidos con la industria en cuestión. En [DD95] podemos encontrar algunos enfoques del problema de corte desde varias áreas de aplicación.

Se entiende que un producto manufacturado es compuesto de piezas menores que pueden ser de diferentes materias, formas, categorias y estilos de textura. Por ejemplo, un calzado puede ser confeccionado por piezas de cuero, piezas de badana, etc. de formas variadas y de diferentes espesuras. Algunas piezas tendrán direcciones de texturas establecidas. En general, cada pieza de un producto manufacturado puede ser cortada de la pieza mayor con ciertas restricciones. Pero también existen productos que no exigen restricciones de orientación específicas, caso de latones, vidrios, cueros

en general, etc [Fa90].

En el intento de resolver las diferentes variantes del problema de cortes se han realizado muchas investigaciones que han dado eficientes resultados, entre ellas tenemos estudios en Programación Lineal [GG65], [GG66], [PGD95], enfoques basados en Heurísticas [MR02], Metaheurísticas [LMP+92], [TCL02], etc.

La cantidad de problemas de cortes y empaquetamiento es tan variada que constituye un área dentro de los problemas de producción [DF92]. Hinxman [Hi80] y Dyckhoff [Dy90] muestran revisiones de estudios realizados, así como clasificaciones de la diversidad de problemas que se presentan. En [DKAG 85] se presenta una clasificación de los problemas de acuerdo a sus características más importantes.

El problema de cortes en dos dimensiones (2D), donde las piezas requeridas se cortan a partir de un planchón o lámina de materia prima, ha sido categorizado, dependiendo de la forma de las piezas a cortar, en dos tipos: *regulares* e *irregulares*. Los cortes regulares son enfocados, generalmente, a obtener cortes de piezas rectangulares; mientras que los cortes irregulares manipulan piezas no rectangulares. Los cortes regulares, que dependiendo de la forma de corte, presentan dos variantes: *guillotinables* y *no guillotinables*.

El problema de cortes regulares guillotinables consiste en determinar la secuencia de cortes de guillotina sobre una superficie rectangular en stock y sus rectángulos resultantes con el objetivo de maximizar la cantidad total de rectángulos producidos [Vi89]. Cada corte es realizado en forma perpendicular entre dos lados paralelos de una superficie. De este modo, cada corte en un rectángulo lo separa en otros dos menores, tal como mostrado por la Figura 2.1(a).

El problema de cortes no guillotinables se caracteriza por no ser corte necesariamente del tipo lineal, no se puede separar un rectangulo en dos partes efectuando un corte de guillotina. El corte no se realiza de extremo a extremo, sino puede realizarse de cualquier forma, tal como mostrado por la Figura 2.1(b). En [BMA02] se muestra en detalle la formulación de este tipo de problema.

Existen muchas estrategias para resolver el problema de cortes regulares. Para cortes tipo Gillotinable se han realizado estudios proponiendo métodos exactos y heurísticos; destacan los trabajos de trabajos de Wang [Wa83], Oliveira y Ferreira [OF90], Parada et al. [PGD95] [PMG95]. El caso no guillotinable también ha sido

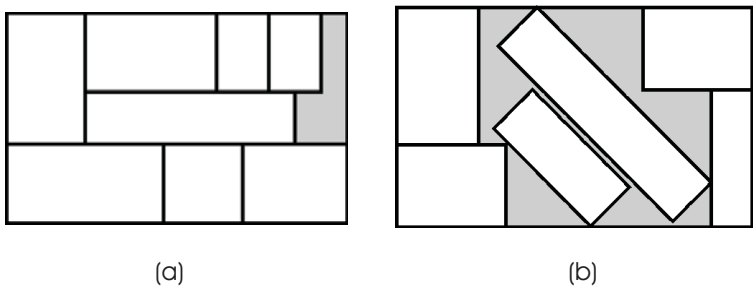


Figura 2.1: Plantillas de cortes regulares: (a) Ortogonal guillotinable; (b) No ortogonal.

ampliamente estudiado, tal como se puede apreciar en los trabajos de Bengtsson [Be82], Biró y Boros [BB84], Beasley [Be85], Daniels y Ghandforoush [DG90].

El problema de corte de piezas regulares se encuentra presente en sectores productivos que manipulan vidrios, maderas, cartones, láminas metálicas, etc. Por ejemplo, existen trabajos realizados en resolver el problema de cortes en madera [Ga96], [VM92], [MG97], en papel [WIH95], [HEIS96], vidrios [DG76], [Fa83], [Ma79], entre otros.

Pero no toda industria usa piezas regulares en sus manufacturas, existen una infinidad de industrias que producen sus productos con piezas irregulares. Refiriéndonos a la geometría de los cortes, podemos encontrar productos como plantillas para zapatos, acabados de muebles, derivados de aluminio, etc. Es así como surge la necesidad de estudiar este tipo de problemas. El problema se caracteriza porque la naturaleza de las piezas a cortar es de cualquier forma geométrica, y se encuentra presente en sectores productivos tales como, confecciones de ropa, artículos de cuero, calzados, construcción de navíos, etc. La Figura 2.2 muestra un ejemplo de casos de cortes irregulares en la industria de confecciones de ropa, mientras la Figura 2.3 muestra un caso típico de piezas de cortes irregulares en la industria de calzados.

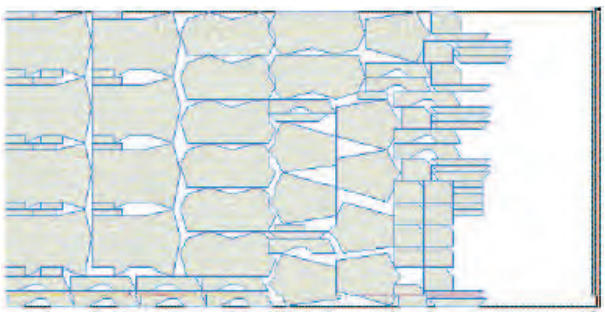


Figura 2.2: Plantilla de corte de piezas en la industria confección de ropas.

Figura 2.3: Plantilla de corte de piezas en la industria de calzados.

En el caso de la industria de confecciones de ropa, como la de cuero, se deben configurar encajes a partir de los moldes de las piezas a cortar. Una vez configurado estos, se procede a cortar tantas piezas de tela como sea necesario para abastecer un determinado requerimiento. Las pérdidas de material no tienen, generalmente, un valor de recuperación significativo para estas empresas, por lo cual no existen mecanismos adecuados para su control.

El método de resolución de problema de cortes ofrece un padrón de corte con las medidas, generalmente en escalas, capturadas de las piezas demandadas, número de cada pieza demandada, y las dimensiones del material disponible a cortarse. Una vez obtenido el padrón de corte, el operario especializado en cortes dispone el material, marca siguiendo las indicaciones del patrón, y ejecuta el corte. En industrias automatizadas, es posible con un brazo de robot haga el papel del operario siguiendo el padrón de corte generado por el resolvedor del problema de corte.

En este trabajo presentamos un modelamiento del problema de cortes irregulares, trabajos existentes que tentan resolver el problema, sus principales variantes y aplicaciones en la industria. Para esto, organizamos el capítulo de la siguiente manera: en la Sección 1 abordamos definiciones y trabajos relacionados con el problema de cortes irregulares, en la Sección 2 mostramos algunos modelos matemáticos que envuelven cortes irregulares, en la Sección 3 revisamos metodologías que han sido aplicadas, en la Sección 4 resumimos algunos aplicativos existentes en la industria, y, por último, en la Sección 5 proponemos un modelo para el problema.

2.1. El Problema de Corte de piezas Irregulares

También conocido en la literatura como *Nesting Problem*, consiste obtener un padrón de corte de un conjunto dado de piezas con irregular forma (típicamente no convexas) a partir de una lámina, que puede ser también de naturaleza irregular, mayor. Esas piezas generalmente son determinadas por molduras pre-definidas; tal es el caso de las piezas para zapatos, que es casi imposible encontrar una pieza rectangular en sus componentes, y la forma de corte es de estilo manzanito, o sea, no guillotinable.

Los trabajos encontrados, con resolución de cortes irregulares, se encuentran en textiles [Fa88], Tapetes [Li77], lona [Fa90], entre otros. En la industria de ropa se destacan los trabajos experimentales de Milenkovic et al. [MDL92] y Li y Milenkovic [LM93], sugiriendo rutinas de mejora en los procesos para producir soluciones manuales, los cuales se consideraban que eran mas eficientes que las soluciones generadas por un computador de última generación. Los estudios de Albano [AL77] y Albano y Sappupo [AS80] describen algoritmos que transforman el problema en encaje de piezas irregulares representados como poligonales. Realmente, todos los trabajos encontrados para problemas de de cortes irregulares se apoyan en objetos de geometria poligonal, que se considera una aproximación bastante eficiente de los objetos irregulares.

2.1.1. Objetos de geometría irregular

Se entiende por objeto de geometría irregular a todo objeto de contorno asimétrico que puede o no tener concavidades. Su geometría puede ser compuesta de pedazos de curvas, segmentos de rectas y puntos.

Según Rivera [Ri01], podemos definir un objeto de geometría irregular a través de la unión de segmentos de curvas, de tamaños variados, descritos por la interpolación de un conjunto de puntos $C = \{c_1, c_2, ..., c_n\}$ distribuidos en las proximidades del contorno del objeto deseado. Cada segmento de curva será obtenido por la interpolapolación de un subconjunto de puntos adjacentes.

Existen varios métodos de interpolación de puntos en el plano, cada una con sus ventajas y sus complejidades y con características propias para cada aplicación.

Para el propósito de nuestro trabajo, usaremos interpolación basado en curvas B-splines cúbicas periódicas cerradas, en su versión parametrizada [RA90]. En este tipo de curvas, los puntos C que se interpolan para generar los segmentos de curva son llamados "puntos de control". En el caso de B-splines cúbicas, cada cuatro puntos de control adyacentes definen un segmento de curva através de la interpolación con coeficientes generados por cuatro funciones cúbicas llamadas bases, tal como muestra la Figura 2.4 (que son simplemente una desplazada respecto a la otra) donde N_1^4, N_2^4, N_3^4 y N_4^4 son las bases en un dominio unitário. La fórmula parametrizada en notación matricial, por ejemplo, para el segmento de curva S_k, está dada por

$$B_k(t) = \frac{1}{6}[t^3\, t^2\, t\, 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}. \tag{2.1}$$

Siendo de esta forma, el segmento de curva S_k definido por

$$S_k(t) = B_k(t)C_k, \text{ para } 0 \leq t \leq 1, \tag{2.2}$$

donde C_k es un vector de cuatro puntos de control adyacentes de C.

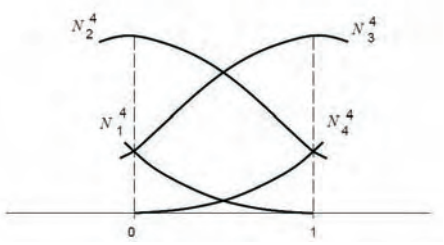


Figura 2.4: *Segmento de funciones-base B-splines cúbica periódicas en un intervalo unitario.*

El objeto deseado será definido por la unión de esos segmentos de curva. Esto es,

$$S = \bigcup_{k=1}^{n} S_k. \tag{2.3}$$

Más detalle sobre la determinación de un objeto se puede encontrar en [Ri01] y [RA90]. La Figura 2.5 ilustra un objeto con sus respectivos puntos de control. La forma de distribución de esos puntos de control definen contornos irregulares, por ejemplo, el posicionamiento bastante próximo de algunos puntos de control definen segmentos que presentan terminaciones con tendencias a puntas agudas; puntos de control colineales

definen segmentos con tendencias a segmentos de rectas, etc. Entonces, cualquier tipo de pieza de corte irregular, en este caso, podrá ser representada por una distribución apropiada de puntos de control.

Figura 2.5: Objeto Irregular compuesto por puntos de control.

2.1.2. Patrón de corte irregular

Antes de iniciarse el procedimiento de corte de la plantilla con las piezas requeridas, lo lógico es que se tenga un patrón de corte para que las piezas requeridas sean debidamente marcadas en la plantilla, de forma que los pedazos intermediarios, que posiblemente serán los desperdicios, sean mínimas. En el caso de cortes regulares, donde el predominio es rectangularidad de las piezas e inclusive la forma de la plantilla, son combinados en función de sus vértices y aristas, uno al lado de otro, cuidando que que no existe la intersección entre las piezas. Algunos trabajos de piezas irregulares aproximados a polígonos, donde se tenga como elementos para la combinación vértices y aristas, son inspirados en métodos de resolución de corte regular, pero con enfoques diferentes que las rectangulares. Siendo así, aún se observa deficiencia en este tipo de problemas.

La idea en los cortes irregulares, se reduce a generación de un patrón de corte el cual presente una distribución eficiente de las piezas irregulares no poligonales en una plantilla que también puede ser irregular. Se busca que no exista intersección entre las piezas distribuidas en la plantilla y que la configuración final tenga una apariencia compactada; de esta forma se pretende reducir los pedazos intermediarios. La Figura 2.6 ilustra un ejemplo de un proceso de compactación para cortes.

La combinación de los objetos con las características definidas como irregular, usando métodos de cortes regulares o los irregulares de constitución poligonal, es sumamente pesada, cuya solución exacta, aún con los computadores actualmente

Figura 2.6: Un ejemplo de compactación de objetos en un sector de una plantilla.

avanzados, es prácticamente imposible de obtenerse en tiempo prudencial. En este sentido, se vale de métodos heurísticos y otros métodos para determinar un padrón de corte considerablemente bueno, del punto de vista de reducir los desperdicios.

2.2. Metodologías

Para resolver problemas de optimización discreta hay diversas metodologías que podemos encontrar en la literatura. Existen métodos exactos los cuales garantizan encontrar la solución óptima a un problema, pero el inconveniente en usarlos es la complejidad computacional, la cual con frecuencia toma mucho tiempo en la resolución de problemas de porte ya que requieren realizar una búsqueda exhaustiva en todo el espacio de búsqueda. Frente a este problema surgieron los métodos heurísticos los cuales raramente encuentran la solución óptima, pero usualmente generan soluciones aceptables en un tiempo corto al tratar problemas de gran porte.

Tenemos también métodos metaheurísticos los cuales han demostrado ser en la mayoría de los casos más eficientes en encontrar una solución aproximada al problema. A continuación, detallaremos algunas metodologías enfocadas a dar solución al problema de cortes irregulares. Algunas de ellas estan basadas en dar solución al problema de cortes regulares las cuales servirán como base para análisis.

2.2.1. Métodos Exactos

Los métodos exactos presentados a continuación estan basados en dar solución al problema de cortes regulares de constitución rectangular.

Gilmore and Gomory [GG65] extendieron su trabajo de cortes unidimensionales al caso de cortes regulares en dos dimensiones, proponiendo un esquema de generación de columnas en el cual se producía una nueva columna cada vez que se realizaba una iteración, resolviendo de esta manera una generalización del problema de Knapsack, en el cual nuevos patrones de corte eran generados. El método planteado por ellos no era recomendable cuando las instancias de prueba eran de gran porte, ya que el proceso de solución podía resultar extremadamente largo. Esto lo señalaron Riehme et al. [RST96] en su estudio. Ellos propusieron un método en el cual trataron el problema de cortes guillotina de dos periodos considerando un número variado de piezas en demanda. En 1966 Gilmore and Gomory [GG66] presentaron un método basado en programación dinámica para resolver tanto el problema de cortes guillotina como el no guillotina. Este método presentaba algunos errores para el caso no guillotinado.

Beasley [Be85] fue el primero en proponer un algoritmo exacto para resolver el problema de cortes no guillotina; este fue basado en el algoritmo de Branch and Bound donde el límite superior fue derivado de la relajación Lagraniana de la formulación del problema en programación entera. Fue usada la optimización subgradiente para optimizar el valor de la cota superior. Los resultados computacionales indicaron que el algoritmo propuesto era capaz de resolver moderadamente problemas de un tamaño significativo. Hadjiconstantinou y Christofides [HC95] desarrollaron también un algoritmo exacto para resolver el problema de cortes no guillotina basado en Branch and Bound; los resultados computacionales demostraron que su algoritmo era competitivo con el método propuesto por Beasley [Be85]. Fekete y Schepers [FS97a] [FS97b] presentaron un árbol de búsqueda de dos niveles para el problema de cortes no guillotina, el cual combina el uso de una especial estructura de datos para una caracterización posible de empaquetamientos con una nueva clase de límites superiores basados en una escala conservativa. El resultado computacional de este método demostró superioridad ante los métodos propuestos por Beasley [Be85] y por Hadjiconstantinou y Christofides [HC95].

Entre otros estudios, usando límites superiores para resolver el problema de cortes

no guillotina, tenemos los trabajos de Tsai et al. [TMM88], Scheithauer [Sc99] y Amaral & Letchford [AL01]. La mayoría de estos trabajos involucra la solución mediante el método de generación de columnas, así como la solucion por el método de knapssack.

Cabe resaltar que estos métodos han servido de base para algunos métodos que solucionan el problema de cortes irregulares de constitución poligonal.

2.2.2. Métodos Heurísticos y Meta-Heurísticos

Los problemas de corte de piezas pertenecen a la clase de problemas para la cual ningún algoritmo polinomial ha sido encontrado, siendo estos problemas de la clase "NP-Difícil". Esto significa que los tiempos computacionales involucrados en la obtención de las soluciones óptimas son demasiado altos y con la tecnología actual no se puede contar con resultados prácticos. Es en estos casos, donde los métodos aproximados juegan un papel de gran importancia, siendo bien justificado el desarrollo de heurísticas para su resolución.

Los métodos heurísticos han sido ampliamente explorados en la resolución del problema de cortes regulares; entre éstos destacan las técnicas de Simulating Annealing [OF93], [Fa99], Grafos AND/OR [MAA92], [PGD95], Tabu Search [LMV99], Algoritmos genéticos [KSV91], [Kr95], [Sm85], Algoritmos híbridos [Ja96], Algoritmo Golosos FFD y BFD [MD02], entre otras. Entretanto, muchos de estos métodos usan directa o indirectamente el proceso de construcción de patrones propuestos por Wang [Wa83].

Algunos de esos métodos han servido de base para los futuros trabajos de reposicionamiento de piezas irregulares de constitución poligonal, los cuales presentaremos a continuación.

Técnicas para resolver el Problema de Cortes Irregulares

En [SBM+97] encontramos una propuesta para la resolución del problema de cortes irregulares no convexos tratando piezas de forma poligonal mediante un algoritmo genético. Para la implementación del algoritmo consideraron colecciones de plantillas de corte potenciales como individuos que conformaban la generación y la función objetivo que encontraron era la suma de las áreas de las piezas posicionadas comple-

tamente en la superficie con respecto al área total utilizada. El algoritmo mostró una eficiencia entre 65-75 % cuando se trataba de piezas de carácter irregular.

Tay et al. [TCL02] presentaron un método basado en algoritmos genéticos para la resolución del problema de cortes irregulares considerando piezas de forma poligonal. El método consistía en el posicionamiento secuencial de las piezas en demanda ayudándose de un algoritmo genético el cual encontraba la mejor posición para ubicar cada pieza a lo largo de los límites de la superficie. El algoritmo propuesto permite posicionar cualquier tipo de pieza en una superficie con el criterio de que la pieza posicionada toque uno de los límites de la superficie. A las piezas se les permitía movimientos de rotación dentro de la superficie para encontrar su mejor ubicación la cual era la función de fitness del algoritmo genético; una vez conseguida la posición, el área de la superficie involucrada era disminuida en el área de la pieza que se esta tratando.

Ismael y Hon [IH92] y Jane et al. [JFR92] abordaron el problema de cortes irregulares poligonales usando algoritmo genético y simulated annealing. Ismael y Hon consideran tan solo piezas de naturaleza lateralmente simétrica y un máximo de 180 grados de rotación para el criterio de posicionamiento de las piezas. El algoritmo que proponen presenta desventaja cuando se enfrenta a piezas que no son de naturaleza lateralmente simétrica. Jane et al. presentan un algoritmo para posicionar diferentes formas de piezas dentro de una superficie, las formas podían ubicarse en cualquier posición. Ellos en su estudio asumen que la posición del punto de referencia sobre la primera pieza posicionada es fijo; para posicionar las siguientes piezas se necesita el ángulo de orientación de la primera pieza, las coordenadas de posicionamiento y orientación de las otras piezas y su desplazamiento dentro de la superficie. Como función de fitness, para cada pieza posicionada se mide la utilización del área, así como la penalidad (costo al superponer una pieza sobre otra al momento de el posicionamiento) si es que se traslapa.

Oliveria y Ferreira [OF93] abordaron el problema de cortes irregulares de forma poligonal. Fijaron el tamaño de una lámina en stock y usaron la técnica de Simulated Annealing para intentar minimizar la longitud. Para la definición del conjunto de posibles soluciones permitieron que las piezas se pudieran traslapar. El costo de su función objetivo es una combinación lineal de la longitud requerida y un término de penalidad para una aproximación a la cantidad mínima de traslapación permitida.

Dowsland y Dowsland [DD93] usaron una formulación similar al problema de Oliveria y Ferreria, pero también fijaron la longitud de la lámina en stock, por lo cual el único costo involucrado era la penalidad por traslapación. Una vez que una posible solución es encontrada, la longitud de la lámina es reducida y luego se intenta el encajamiento de la plantilla de corte dentro de la nueva longitud.

Blazewicz et al. [BHW93] sugirió un algoritmo de Tabu Search en el cual no se permite traslapación de piezas en las plantillas de corte. Las soluciónes son obtenidas moviendo las piezas hacia una nueva posición buscando siempre un mejor movimiento en vez de coger una pieza aleatoria para posicionar. Este enfoque es superior que algunos métodos estándar de posicionamiento como el de Albano y Suppopo [AS80], pero algunas veces presenta desventajas, ya que pierde soluciónes obvias, las cuales sí son encontradas por otros métodos.

Milenkovic [Mi88] usa la simulación física basada en posicionamiento compactar objetos de forma poligonal representando cortes de ropa, obteniendo resultados relativamente buenos.

Dowsland y Dowsland [DD02] propusieron una rápida y eficiente implementación de un algoritmo de posicionamiento Bottom-Left para el empacamiento de polígonos. El algoritmo permite ir posicionando piezas en una superficie temporal con lo cual garantiza llenar espacios vacíos al momento de ir consiguiendo soluciones, permitiendo tener un conjunto infinito de soluciones para obtener el posicionamiento adecuado de cada una de las piezas a colocar. Su propuesta muestra alta confiabilidad en el manejo de posicionamiento de piezas poligonales, con lo cual se puede extender este estudio hacia algoritmos más avanzados que proveen criterios de selección de piezas.

Bennell et al. [BDD01] proponen una técnica denominada “NoFit Polygon”, la cual es una poderosa herramienta que trata la geometría de piezas irregulares de constitución poligonal para construir un algoritmo que resuelva el problema de cortes irregulares. El NoFit Polygon es obtenido a partir de dos polígonos, calculándose el más pequeño límite convexo para estos dos polígonos; este proceso se repite para todos los polígonos a tratar. Para no caer en un cuello de botella, proponen la implementación de una cache, la cual guarda la información de evaluaciones previas de soluciones que ya han sido evaluadas. En su propuesta demostraron que aumentando el tamaño de la cache la velocidad de procesamiento de soluciones del algoritmo se incrementa considerablemente. Gomes y Oliveira [GO01] desarrollaron un algorit-

mo GRASP para resolver el problema de cortes irregulares considerando piezas de naturaleza poligonal y una superficie contenedora de naturaleza rectangular (ancho fijo e infinito largo). El objetivo del algoritmo era la minimización de la longitud de la superficie, buscando una secuencia de piezas para un adecuado posicionamiento. Para conseguir ello utilizaron una heurística de posicionamiento "Bottom-Left"para la generación de los patrones de corte.

Kim and Pellacini [KP02] introducen una técnica llamada Jigsaw image Mosaics para la construcción de imágenes finales a partir de formas arbitrarias. Teniendo en cuenta que tanto el contenedor como las figuras son de forma arbitraria, se desea llenar el contenedor hasta llegar a su compactación deformando ligeramente las piezas para un mayor efecto visual. Introducen un framework basado en criterios de energía donde la construcción de un mosaico es definido en términos de energía, permitiendo la creación de varios mosaicos con tan solo cambiar la función de energía. El algoritmo propuesto dio buenos resultados en la construcción de mosaicos, con lo cual se puede extender a atacar problemas de empacamiento de objetos usando técnicas más avanzadas como Simulated Annealing gracias a la función de energía con la cual trabaja el algoritmo.

2.2.3. Comparación de los métodos de solución

Comparar los métodos de solución que hemos tratado anteriormente resulta difícil en un sentido cuantitativo. Los tipos de problemas estudiados difieren uno del otro no habiendo a primera vista comparación relevante acerca de los métodos que se han aplicado. También tenemos que tomar en cuenta que los métodos han sido aplicados en diferentes ambientes, significando esto que los tiempos de CPU tomados para obtener las soluciones no pueden ser comparados. Sin embargo, las comparaciones han sido realizadas tomando en cuenta diferentes subconjuntos de métodos de solución. En la Tabla 2.1 son presentadas las comparaciones entre los métodos de solución.

Tabu Search parece producir soluciones más exactas para el problema, pero converge más lento. Tomando en cuenta los resultados obtenidos por los métodos estudiados, estos pueden ser bien considerados como Ant Colonization, Programación Evolutiva y Algoritmos Genéticos de acuerdo al grado de exactitud en brindar la solución; así como de acuerdo a la velocidad de convergencia [DL02]. Si la veloci-

dad de convergencia es considerada más importante que la exactitud de la solución, las heurísticas y metaheurísticas pueden ser mejores alternativas. El mayor inconveniente de los métodos heurísticos es que pueden caer atrapados en un óptimo local. Los métodos metaheurísticos pueden evitar caer en esto, escogiendo soluciones candidatas fuera del espacio de búsqueda local. En los métodos metaheurísticos, el proceso de solución es con frecuencia guiado por alguna heurística de bajo nivel como lo vimos anteriormente, formando métodos híbridos.

Los métodos algorítmicos solo proveen el óptimo global para el problema. Sin embargo el costo de exactitud debe ser pagado por el lento tiempo de convergencia.

Métodos existentes		
Método	**Ventajas**	**Desventajas**
Programación Lineal	Muy eficiente cuando el tamaño del problema es moderado, las reglas para los patrones son seteadas fácilmente.	Tiempo computacional costoso cuando se enfrenta a resolver patrones de corte de gran porte.
Tabu Search	Produce altas soluciones de calidad en comparación al GRASP.	Converge más lento que el GRASP.
GRASP	Muy rápido en comparación a Tabu Search. Produce soluciones fiables.	Produce menos soluciones exactas que Tabu Search.
Programación evolutiva	Simple, rápido y más exacto que un algoritmo genético.	Pierde exactitud cuando se enfrenta a problemas grandes.
Algoritmo Genético	Relativamente simple para implementación.	Más lento en convergencia que la Programación evolutiva.
Simulated Annealing	Produce soluciones de calidad en comparación al GRASP.	Converge más lento que Tabu Search o GRASP.

Cuadro 2.1: Tabla comparativa de los métodos existentes para resolver el problema de cortes

2.3. Aplicativos Existentes

Un número considerable de aplicaciones han sido desarrolladas para brindar asistencia a la creación de patrones de cortes adecuados dando solución al problema de cortes y empacamiento, considerando objetos de diversos materiales. En la Tabla 2.2 se muestran aplicaciones comerciales que dan solución en cierta medida al problema. Muchos de estos aplicativos están disponibles en versiones demo, por lo cual resulta

dificultoso obtener información acerca de los métodos de solución aplicados. A continuación procederemos a explicar algunos aplicativos de los cuales hemos encontrado información acerca del método de solución en la cual han sido implementados.

2NA es desarrollado por Boeing, ellos utilizan para su solución métodos aplicados a cortes rectangulares, el aplicativo es rápido para el problema de empacamiento de stencil por el cual fue hecho, pero puede ser extendido hacia otros materiales. AutoNester está basado en el trabajo realizado por Heistermann and Lengauer [HL95]. Presenta dos librerías AutoNester-T (resuelve el problema de cortes irregulares en la industria textil) y AutoNester-L (resuelve el problema de cortes irregulares en la industria de cuero) las cuales presentan dos métodos de solución. AutoNester-T se encuentra basado en Simulated Annealig el cual da velocidad y calidad para conseguir las soluciones.

Aplicativos	
Nombre	**Homepage**
2NA	http://www.boeing.com/phantom/2NA/
AutoNester	http://www.gmd.de/SCAI/opt/products/index.html
NESTER	http://www.nestersoftware.com/
Nestlib	http://www.geometricsoftware.com/geometry_ct_nestlib.htm
OPTIMIZER	http://www.samtecsoft.com/anymain.htm
PLUS2D	http://www.nirvanatec.com/nesting_software.html
MOST 2D	http://www.most2d.com/main.htm
Pronest/Turbonest	http://www.mtc-limited.com/products.html
SigmaNEST	http://www.sigmanest.com/
SS-Nest/QuickNest	http://www.striker-systems.com/ssnest/proddesc.htm

Cuadro 2.2: Aplicativos existentes para resolver el problema de cortes.

2.4. Modelo del Problema

Como se mencionó anteriormente, el problema de cortes irregulares que estamos analizando esta definido por objetos de forma irregular, cuyos contornos están constituidos por segmentos de curvas B-splines cerrados, presentando áreas calculables usando métodos numéricos relativamente simples. Así mismo, la superficie a ser cortada será definida de la misma forma, la cual suele de geometria irregular, por ejemplo un planchón de cuero.

Para el entendimiento del modelo que presentaremos a continuación llamamos "superficie contenedora" a la representación de la plancha a ser cortada, así como las piezas a cortar serán los objetos de geometría irregular, los cuales presentan una cantidad determinada de demandas de corte.

Son dos objetivos los que perseguimos con la formulación del presente modelo. El primero se refiere a conseguir que la distribución de las piezas en demanda en una superficie contenedora ocupe menor area posible, y el segundo está referido al uso del menor número de superficies contenedoras para las piezas en demanda.

Consideremos m piezas definidas por segmentos de curvas B-splines cúbicas. Se parte teniendo un conjunto de piezas en demanda $p_j \in P$, las cuales poseen un área $Areap_j$ para $j = \{1, .., m\}$. Estas piezas deben ser distribuidas en una superficie contenedora Y de área $Area(Y)$.

A continuación procedemos a identificar las partes del modelo a realizar.

- **Criterio:** Minimizar el número de contenedores
- **Variables de Decisión:**

$$Y_i = \begin{cases} 1 & \text{si se usa la superficie } i \\ 0 & \text{en caso contrario.} \end{cases}$$

$$X_{ij} = \begin{cases} 1 & \text{si la pieza } j \text{ es seleccionada en la superficie } i \\ 0 & \text{en caso contrario.} \end{cases}$$

- **Función Objetivo:** $\sum_{i=1}^{n} Y_i$
- **Restricción:**

  (a) Capacidad: la suma de las areas p_j de solo aquellas piezas X_{ij} que ocupan el contendor Y_i debe ser menor que el area de del contenedor Y_i,

$$\sum_{j=1}^{n} X_{ij} Area(p_j) \leq Area(Y) Y_i, \quad \forall i \in \{1, .., n\}$$

  (b) Unicidad: un requerimiento X_{ij} solo puede ser posicionado una sola vez en una superficie contenedora Y_i,

$$\sum_{i=1}^{n} X_{ij} = 1, \quad \forall j \in \{1, .., n\}$$

(c) Exclusión: la pieza p_r no puede intersectar a la pieza p_s

$$p_r \cap p_s = \emptyset, \quad \forall r \neq s, 1 \leq r, s \leq n$$

El modelo queda de la siguiente manera:

$$\begin{array}{ll} \text{Minimizar} & \sum_{i=1}^{n} Y_i \\ \text{s.a} & \text{(a) Capacidad} \\ & \qquad \sum_{j=1}^{n} X_{ij} Area(p_j) \leq Area(Y) Y_i, \quad \forall i \in \{1, .., n\} \\ & \text{(b) Unicidad (requirimiento } j \text{ respecto a contenedor } i) \\ & \qquad \sum_{i=1}^{n} X_{ij} = 1, \quad \forall j \in \{1, .., n\} \\ & \text{(c) Exclusión} \\ & \qquad p_r \cap p_s = \emptyset, \quad \forall r \neq s, 1 \leq r, s \leq n \\ & \text{donde } x_{ij} \in B = \{0, 1\}. \end{array} \tag{2.4}$$

Capítulo 3

Arquitectura del Sistema

El presente capítulo aborda la arquitectura del sistema para resolver el problema de cortes irregulares. Esta arquitectura se encuentra dividida en cuatro componentes principales los cuales son: Módulo de Entrada, Módulo Metaheurística de Selección, Módulo de Compactación por Simulación Dinámica y Módulo de Salida. Cada sección del presente capítulo describira cada módulo, así como también se describira en una sección el funcionamiento del sistema.

El módulo de entrada se encarga de obtener la información de los datos ingresados por el usuario. Esta información se almacenará en una estructura de datos donde se alojen las características de las piezas y las superficies irregulares a procesar. Asimismo este módulo servirá como interfase para la comunicación con el usuario.

El módulo metaheurística de selección determina el número de contenedores a utilizar y la lista de piezas a colocar en cada contenedor para una determinada cantidad de piezas en demanda.

El módulo de compactación por simulación dinámica realiza la compactacion de las piezas en sus respectivos contenedores, consiguiendo una adecuada configuración final de las piezas en demanda y una minimización de los contenedores que las alojan.

El módulo de salida realiza la presentación de los resultados de la ejecución del algoritmo propuesto.

En la Figura 3.1 presentamos un diagrama de la arquitectura del sistema de cortes irregulares, presentado los tres módulos anteriormente mencionados.

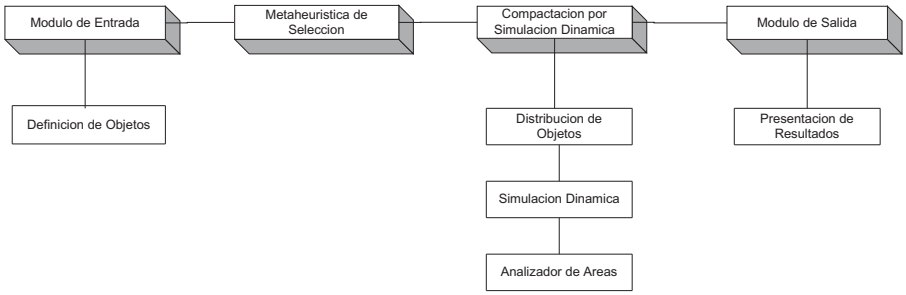


Figura 3.1: Diagrama de Arquitectura del Sistema.

3.1. Módulo de Entrada

La adquisición de datos será realizada por medio de una interfase usuario la cual permitirá el ingreso de las características geométricas, dinámicas y de simulación de las piezas y superficies a procesar. Asimismo realizará el ingreso de la configuración de parámetros necesarios para el funcionamiento del algoritmo de procesamiento de piezas irregulares.

3.1.1. Definición de Objetos

Una vez que la información de los objetos es ingresada al sistema, se almacenará en estructuras de datos fijas o temporales las cuales albergarán las características particulares de cada objeto. Estas estructuras serán utilizadas para representar adecuadamente las características de cada objeto. Para las superficies contenedoras (las cuales pueden ser de diferente tamaño y forma) y las piezas irregulares dichas estructuras almacenarán el identificador del objeto, el número y los puntos de control del que esta compuesto cada objeto, así como sus características dinámicas tales como centro de masa, momentos de inercia, velocidad angular, orientación, etc.

Los datos generales que permiten definir la configuración de los parámetros para el procesamiento del algoritmo así como la definición del ambiente de simulación son el número de objetos, tamaño fijo de paso de tiempo, máximo error permitido en cada paso de tiempo, tolerancias que permiten el control de movimientos, así como otras características visuales.

3.2. Módulo Metaheurística de Selección

Este módulo es el encargado de crear una lista de superficies las cuales contienen piezas irregulares que conformaran los patrones de corte. El algoritmo que invocará este modulo construye patrones de corte a partir de una lista de n piezas, donde va extrayendo subconjuntos de m piezas para satisfacer las demandas de cada contenedor con el objetivo de minimizar el número de contenedores utilizados. La Figura 3.2 muestra un ejemplo del funcionamiento del módulo.

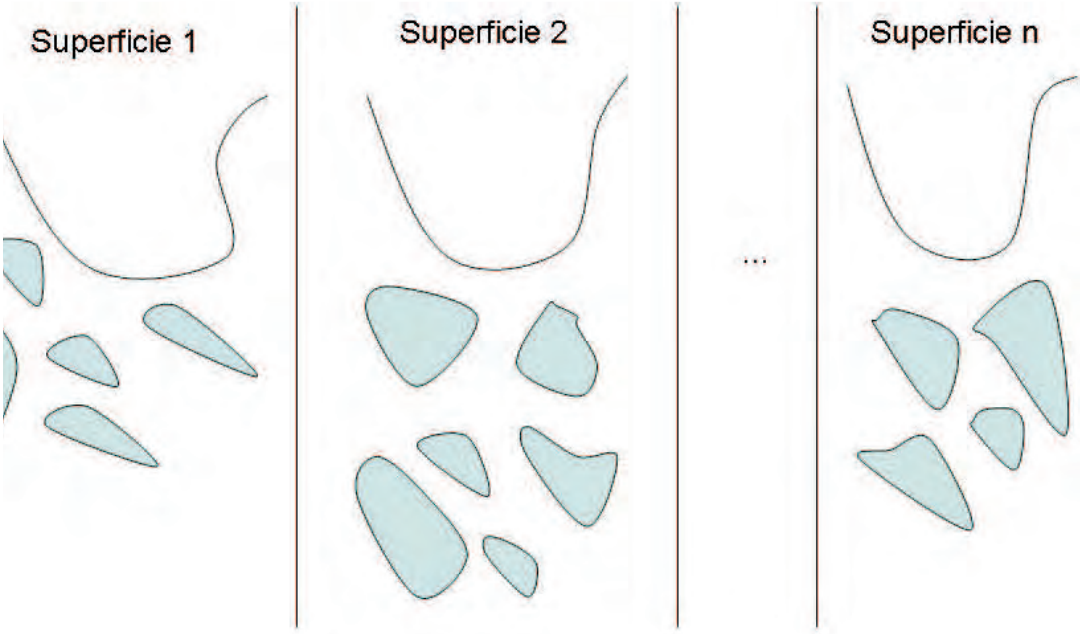


Figura 3.2: Ejemplo de Metaheurística de Selección.

3.3. Módulo de Compactación por Simulación Dinámica

El módulo de compactación por simulación dinámica es el responsable de la coordinación del funcionamiento del algoritmo el cual se inicia con la distribución de los objetos en sus respectivos contenedores, compactación y llegando hasta el analizador de áreas. Después de tener definido las estructuras de datos que albergan las características de cada objeto (piezas y superficies irregulares), así como definido un

escenario apropiado para la simulación se procede a realizar una iteración de soluciones la cual es controlada por un número máximo de iteraciones.

Este módulo esta compuesto por tres submódulos los cuales son distribución de objetos, el cual arma la configuración inicial de los objetos seleccionados; el módulo de simulación dinámica el cual es encargado de la compactación de los objetos y el módulo de análisis de áreas que realiza el cálculo del área final ocupada por las piezas compactadas.

3.3.1. Distribución de Objetos

Este módulo es el encargado de la creación del estado inicial (actual) de los objetos para luego proceder a su compactación. La distribución de las piezas se realizará de acuerdo al resultado de la selección de las mismas en las superficies contenedoras. Se mostrará cada contenedor con sus respectivas piezas irregulares, siguiendo una distribución inicial óptima la cual será realizada por una heurística que muestre la mejor distribución de las piezas seleccionadas para posicionarlas dentro de su superficie. La Figura 3.3 muestra una distribución de piezas en forma vertical, para mas detalle revisar el Capítulo 5.

En primera instancia, se tomará un posicionamiento de piezas en forma vertical; ya que al posicionar las piezas una tras otra de esta manera, sobre sus respectivos contenedores se pretende que el encajonamiento de las mismas se haga imitando la forma que la naturaleza posee para compactar un cuerpo considerando diversos parámetros como el de gravedad.

La configuración inicial de las piezas involucra que cada objeto tenga un estado, que muestre su representación; el cual consiste en la cuantificación de la dinámica, posición y orientación en cada instante de tiempo. Para generar el movimiento asociado a cada cuerpo se van a realizando cálculos a todos los elementos dinámicos de cada cuerpo. Estos cálculos se realizarán considerando las restricciones impuestas a cada objeto. También serán calculadas las orientaciones de los objetos en cada instante así como la transformación de las coordenadas de los mismos.

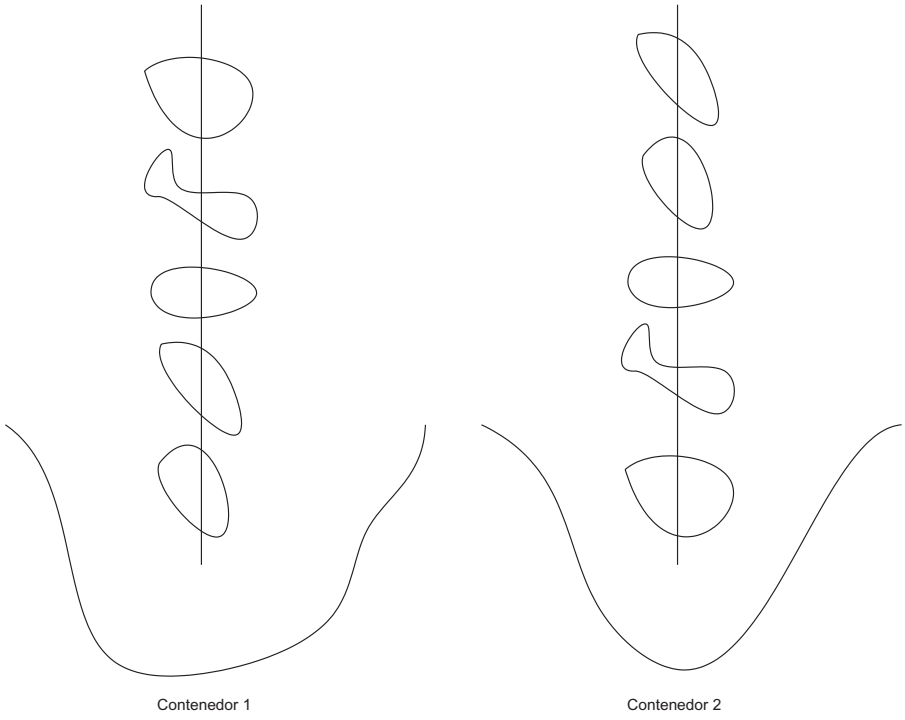


Figura 3.3: Ejemplo de Distribución de Objetos.

3.3.2. Simulación Dinámica

Este módulo se encarga de la generación de movimiento el cual es ejecutado en cada intervalo de tiempo llamado "paso de tiempo". Aquí serán generados los cuadros o frames los cuales realizan la animación de los objetos, siguiendo un comportamiento dinámico.

Las piezas en el ambiente de simulación se encuentra inmersas en el campo gravitacional actuando en dirección hacia en fondo de la superficie contenedora adquiriendo comportamientos dinámicos e interacción con las otras piezas y las paredes de la superficie que las contiene. Los cuerpos se van moviendo en este espacio hasta llegar a un estado final casi estático llamado estado de compactación [Ri01].

Para generar movimiento se van realizando ciclos incrementando el paso de tiempo. En cada iteración se obtiene el estado actual de los cuerpos, se realizan cálculos invocando subsistemas dinámicos los cuales permiten generar nuevas posiciones, velocidades y aceleraciones de los objetos [Ri01]. En cada paso de tiempo se va verificando la existencia de eventos excepcionales así como errores de cálculo. La manipulación adecuada de estos eventos excepcionales, permiten la detección de contactos, interfe-

rencias, tratamiento de colisiones, etc. Para más detalle revisar el capitulo 6 en el cual se hace un análisis de la simulación dinámica de los cuerpos. La Figura 3.4 muestra un estado de compactación de los cuerpos.

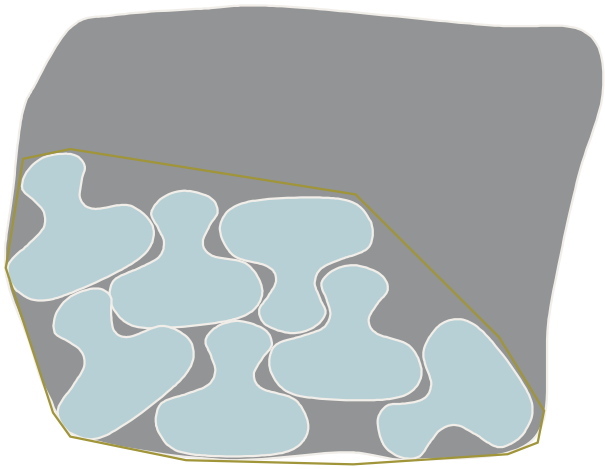


Figura 3.4: Ejemplo de Simulación Dinámica.

3.3.3. Analizador de Areas

Este módulo realiza el cálculo de las áreas finales de los objetos luego de realizada la compactación de los mismos (estado final) en el ambiente de simulación. Se procede a calcular un cerco convexo el cual envuelva a las piezas compactadas mediante un algoritmo de la geometría computacional. La Figura 3.5 muestra el cerco convexo para un conjunto de piezas compactadas.

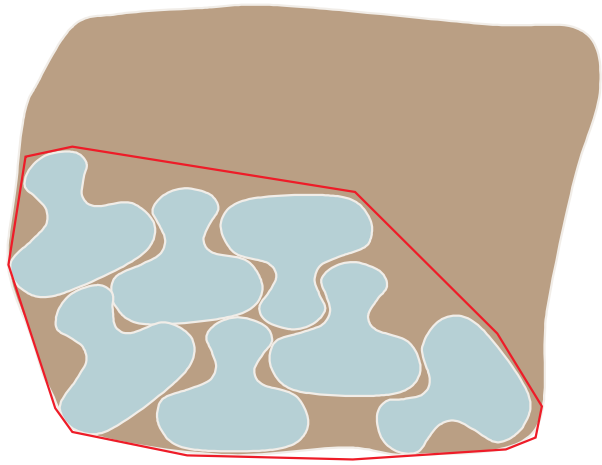


Figura 3.5: Ejemplo de Analizador de Areas.

De esta manera el área total desperdiciada se calcularía restando el área de la superficie contenedora con el área del cerco convexo de las piezas. Para más detalle se puede revisar el Capitulo 7 donde se analiza el cerco convexo para un conjunto de piezas.

3.4. Módulo de Salida

Este módulo se encarga de mostrar los resultados obtenidos por el algoritmo. Existe un proceso de visualización que mostrará los patrones de corte resultantes permitiendo ver a los objetos involucrados en el ambiente de simulación en forma real. Se realizará para este fin un proceso de conversión de coordenadas de un sistema loca a un sistema global.

3.4.1. Presentación de Resultados

Muestra el número de superficies a utilizar, los patrones de corte involucrados en cada superficie, el área de utilización total, el area de utilización de cada patrón de corte y la configuración final de las piezas inmersas en cada patron de corte.

3.5. Funcionamiento del Sistema

Inicialmente, el modulo de entrada realiza la captura de datos obteniendo las características de las piezas y superficies involucradas. Luego el modulo de Metaheurística de selección realiza una selección adecuada de piezas para posicionarlas en un número determinado de superficies contenedoras, aqui el algoritmo entra en un bucle, donde iterativamente, mientras no se cumpla una condición de parada se invoca al módulo de compactación por simulación dinámica. Este módulo permitirá conseguir una configuración final óptima de las piezas en demanda. Para éste propósito el módulo de compactacioń por simulación dinámica utiliza el submódulo de distribución de objetos donde se realizará la configuración inicial de las piezas en sus determinadas superficies. Seguidamente, el submódulo de simulación dinámica realizará la compactación de las piezas hasta conseguir un estado final de las mismas. Finalmente, el submódulo

de análisis de áreas realiza el cálculo de las áreas finales de la solución local obtenida.

Si se cumple la condición de parada se invoca al módulo de salida, el cual mostrará la mejor solución encontrada por el algoritmo con sus patrones de corte respectivos.

La Figura 3.6 a continuación presenta un diagrama de flujo del funcionamiento del sistema de cortes irregulares poligonales:

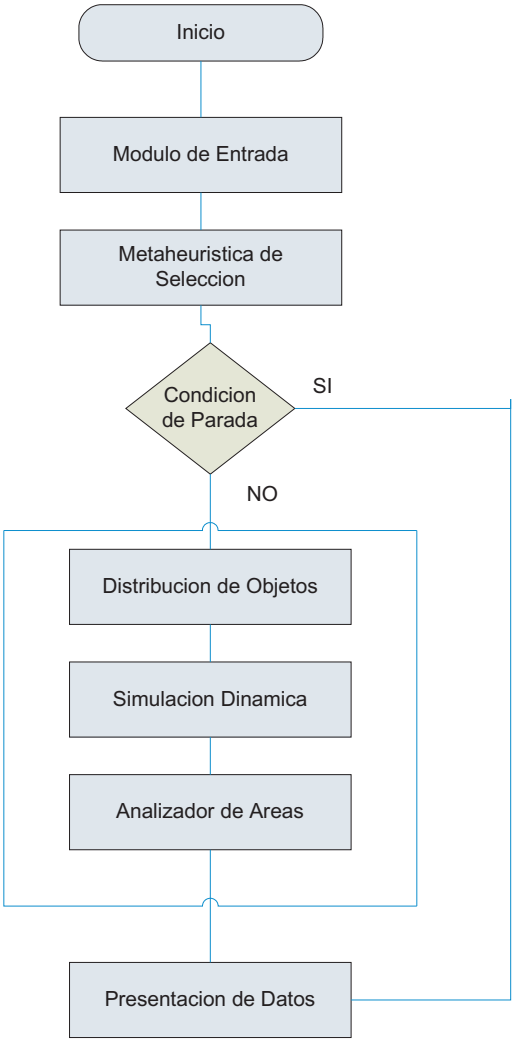


Figura 3.6: Diagrama de Flujo de la Arquitectura del Sistema.

Capítulo 4

Un Algoritmo GRASP para Resolver el Problema de Cortes Irregulares

El capítulo esta organizado de la siguiente manera: en la Sección 1 entraremos a ver la estructura del algoritmo GRASP y en la Sección 2 mostraremos la construcción del algoritmo para resolver el problema de cortes irregulares.

4.1. Algoritmo GRASP

La metaheurística GRASP se caracteriza por ser un procedimiento iterativo que combina una fase constructiva y una fase de mejoría. En la fase constructiva una solución es construida paso a paso adicionando elementos a una solución parcial. En vez de escoger el elemento a ser adicionado, presenta una función golosa, la cual es dinámicamente adaptada como la solución parcial para luego ser construida. Sin embargo, la selección de un elemento no es determinístico, pero esta sujeto a un proceso aleatorio. En ese sentido, cuando repetimos el proceso podemos obtener diferentes soluciones. Después de cada fase constructiva, la fase de mejoría, que usualmente consiste en una búsqueda local simple, trata de sustituir algunos elementos de la solución, los cuales son resultado de una aleatoriedad, produciendo mejores soluciones globales.

El método GRASP presenta dos principales fases. La primera fase se refiere a la

construcción goloso - aleatorio de una solución y la segunda fase a la mejora de la solución construida. En cada iteración GRASP se va registrando la mejor solución encontrada. El algoritmo termina cuando alguna condición de parada es verificada. La calidad de la solución depende tanto del parámetro de relajación α y de la condición de parada. Estas afirmaciones serán observadas al momento que realizamos la corrida de los casos de prueba para el algoritmo.

Para el propósito del presente trabajo hemos realizado un algoritmo GRASP con 2 parámetros de relajación α y β. La fase constructiva esta basada en un procedimiento $FFDConstruccionGRASP$ el cual construye una solución formando patrones de corte, para luego replicarlos de acuerdo a la cantidad de demandas que atienden esos patrones.

4.1.1. Fase de Construcción

La fase de construcción de un algoritmo GRASP incorpora características golosas, aleatorias y de adaptabilidad. El criterio goloso es alcanzado en cada iteración de la fase de construcción rankeando todos los elementos disponibles en una lista de candidatos. Esta es llamada lista de candidatos restrictos (RCL) la cual es luego restringida solo a los elementos que conducen a buenas soluciones. Seleccionando aleatoriamente un elemento de la lista de candidatos restrictos, se da al algoritmo la característica de aleatoriedad en la cual diferentes soluciones pueden ser obtenidas. Finalmente la adaptabilidad es alcanzada por actualización al momento en que en cada iteración hacemos una comparación para poder actualizar la mejor solución con la solución de la iteración anterior (la función golosa puede variar en cada iteración de la fase de construcción).

En el presente trabajo de tesis realizamos la construcción de la lista de candidatos restrictos tomando en cuenta una función golosa que es el área de las piezas en demanda.

Sea $Area(p_k)$ el área de la pieza p_k; se obtiene la mínima y máxima área $Area_{min} = Min\{Area(p_k) : p_k \epsilon E\}$ $Area_{max} = Max\{Area(p_k) : p_k \epsilon E\}$ con la cual se construye la lista de candidatos restrictos (RCL), tomando en cuenta el criterio goloso de maximizar área de las piezas que se encuentran en demanda. El RCL obtenido es $RCL = \{p_k \epsilon E : Area_{max} - \alpha(area_{max} - area_{min}) \leq Area(p_k) \leq Area_{max}\}$, donde E

es la lista de piezas en demanda y $\alpha(0 < \alpha < 1)$ es el parámetro de relajación que controla la restricción de la lista de piezas candidatas imponiendo un valor de umbral sobre la distancia a la mejor área encontrada (realmente α es un porcentaje entre la peor y la mejor área encontrada). De esta manera podemos cambiar el algoritmo desde ser completamente goloso $\alpha = 0$ hasta ser completamente aleatorio $\alpha = 1$.

Una vez construido el RCL, se procede a escoger aleatoriamente una pieza $P_r : \{P_r \epsilon RCL\}$ la cual será encajada a un contenedor S.

4.1.2. Actualización de la Solución

Mientras se va iterando, el algoritmo va comparando la solución obtenida en la iteración con la mejor solución. Si la solución obtenida en la iteración es mejor que la solución actual, se actualiza la solución. En el problema que estamos analizando el criterio para actualizar la solución es el número de contenedores; si la solución que se va encontrando usa menos número de contenedores, entonces se actualiza la solución.

4.1.3. Condición de Parada

Para el problema que estamos analizando utilizaremos los siguientes criterios de parada:

- **Máximo Número de Generaciones:** El algoritmo va iterando teniendo en cuenta el máximo número de generaciones permitidas, con lo cual si se alcanza este número el algoritmo para y se obtiene una solución la cual será la óptima para esa generación.
- **Convergencia del Algoritmo:** Está conformada por un parámetro de número de generaciones en las cuales se consigue una solución local óptima. El algoritmo va iterando generación tras generación y consiguiendo una solución cada vez mejor. Se va comparando la solución local óptima obtenida hasta ese momento con la mejor solución de la generación que se está analizando. Si esta solución es mejor, se reemplaza.

El algoritmo termina si es que la solución local óptima no ha cambiado luego de un número de iteraciones igual al número de generaciones de convergencia ingresadas como parámetro.

4.2. Construcción del algoritmo

A continuación se muestra el diseño del algoritmo GRASP para resolver el problema de cortes irregulares; teniendo en cuenta todos los puntos que hemos tratado anteriormente. Se desarrollará el programa principal y cada uno de los módulos que lo conforman, mostrando su funcionamiento y pseudocódigo en cada parte.

En el cuadro 4.1 se muestra una descripción de las variables a utilizar.

Variable	Descripción
Niteraciones	Número de iteraciones que se están tratando
α	Parámetro de relajación lista de candidatos restrictos (RCL)
β	Parámetro de relajación para determinar la cantidad de piezas
ListaSuperficies	Lista con Superficies encontradas por el GRASP

Cuadro 4.1: Variables del sistema

4.2.1. Programa Principal

Se leen las instancias de entrada; se inicializan las variables $mejorSolucion$ y $menorNumeroSuperficies$; se entra a un bucle verificando la condición de terminación, mientras ésta no se cumpla se construye una solución mediante el procedimiento $FFDConstruccionGRASP$, se realiza el cálculo de las áreas de las piezas involucradas en la solución encontrada mediante el procedimiento $CalcularAreasFinales$, a continuación se compara la solución obtenida con la solución optima local obtenida hasta el momento; procediendo a su actualización si la presente solución es mejor. El respectivo algoritmo, en notación de pseudocódigo, es como sigue:

1: Leer$(n, p_1, ..., p_n, area(p_1), ..., area(p_n), Demanda(p_1), ..., Demanda(p_n), nIteraciones, \alpha, \beta)$
2: $mejorSolucion \leftarrow \emptyset$
3: $menorNumeroSuperficies \leftarrow +\infty$
4: **while** $NOTCondicionParada()$ **do**

5: $listaSuperficies \leftarrow FFDConstruccionGRASP$
6: $utilizacion \leftarrow CalcularAreasFinales(listaSuperficies)$
7: $superficiesusadas \leftarrow tamano(listaSuperficies)$
8: $ActualizarSolucion(superficiesusadas, menornumerosuperficies)$
9: **end while**

Comentando el algoritmo, en el paso 5 se realiza la construcción de una solución. La variable $listaSuperficies$ contiene una lista con un número de superficies contenedoras las cuales indican los patrones de corte a realizar. En el paso 6 la variable $utilizacion$ indica el porcentaje de utilización de material determinado por la solución que se esta analizando. En el punto 8 se realiza la actualización de la mejor solución encontrada.

4.2.2. Leer instancias de entrada

Se definen los parámetros del algoritmo: parámetro de relajación α, parámetro de relajación β, numero de iteraciones $Niteraciones$

Se definen los parámetros requeridos en la demanda de corte: Lista de piezas irregulares en demanda $E = \{p_1, p_2, ..., p_n\}$, Cantidad de Requerimientos de Corte $D = \{Demanda(p_1), .., Demanda(p_n)\}$, área de cada pieza irregular $A = \{Area(p_1), .., Area(p_n)\}$.

4.2.3. FFDConstruccionGRASP

Este procedimiento construye una lista de superficies las cuales contienen patrones de corte.

Para su funcionamiento, se procede a inicializar las variables $listaSuperficies \leftarrow \emptyset$, $i \leftarrow 1$. Se entra en un bucle hasta que se terminen las piezas en demanda repitiéndose iterativamente lo siguiente:

- Se realiza la construcción de un patrón de corte mediante el uso del procedimiento $ConstruccionGRASPFFDContenedor(i, WE)$

- Se realiza la configuración inicial de las piezas en la superficie usando el procedimiento $ConfigurarPiezas(S_i)$
- Se realiza la compactación de las piezas involucradas en el patrón hallado S_i mediante el uso del procedimiento $SimulacionDinamica(S_i)$
- Se realiza el reacomodo de las piezas compactadas mediante el uso del procedimiento $RealizarAjusteSacudida(S_i)$
- Se realiza el calculo de las áreas finales involucradas al patrón hallado S_i mediante el uso del procedimiento $CalcularAreasFinales(S_i)$
- Se realiza un reacomodo de piezas verificando si es posible hacer mas encaje
- Se replica el patrón de corte hallado S_i,se actualizan las demandas, se agrega el patrón a la lista de superficies y se actualiza la lista E de piezas atendidas E $E = E - PiezasAtendidas(S_i)$

El algoritmo en pseudocódigo correspondiente es como sigue:

1: $E \leftarrow \{p_1, p_2, ..., p_n\}$ {contiene la lista de piezas a atender}
2: $listaSuperficies \leftarrow \emptyset$
3: $i \leftarrow 1$
4: **while** $listaPiezas \neq \emptyset$ **do**
5: $WE \leftarrow E$ {WE es la lista de piezas temporal en demanda}
6: $S_i \leftarrow ConstruccionGRASPFFDContenedor(i, WE)$ {contruye un patrón de corte}
7: $ConfigurarPiezas(S_i)$ {realiza la colocacion de las piezas en la superficie}
8: $SimulacionDinamica(S_i)$ {realiza la compactacion}
9: $RealizaAjusteSacudida(S_i)$ {reacomoda piezas compactadas}
10: $CalcularAreasFinales(S_i)$
11: **for** $y = 1$ to $niterConvergencia$ **do**
12: $WE' \leftarrow PiezasAtendidas(S_i)$ {Se obtiene la lista de piezas atendidas}
13: $WE \leftarrow E - WE'$
14: **while** $((AreaDisponible(S_nsup) > Min\{Area(p_r) : r \in WE\}) \wedge (WE \neq \emptyset))$ **do**
15: $minarea \leftarrow Min\{Area(p_r) : r \in WE\}$
16: $maxarea \leftarrow Min\{Max\{Area(p_r) : r \in WE\}, AreaDisponible(S_{nSup})\}$

17: $RCL \leftarrow \{r \in E : maxarea - \alpha(maxarea - minarea) \leq Area(p_r) \leq maxarea\}$

18: $k \leftarrow Random(RCL)$

19: $u \leftarrow Min\{\lfloor \frac{AreaDisponible(S_{nSup})}{Area(p_k)} \rfloor, Demanda(p_k)\}$ {cantidad de piezas que se pueden atender}

20: $umax \leftarrow u$

21: $umin \leftarrow umax - \beta * umax$

22: $cantidadreplica \leftarrow Random([umin, umax])$

23: $EncajarPieza(p_k, cantidadReplica, RCL)$ {Se encaja la pieza}

24: $WE \leftarrow WE - p_k$

25: **end while**

26: $SimulacionDinamica(S_i)$

27: **end for**

28: $nrepPatron \leftarrow \frac{Min}{1 \leq b \leq n}\{\lfloor \frac{Demanda(p_b)}{NumPiezasAtendidas(p_b)} \rfloor : numPiezasAtendidas > 0\}$

29: **for** $t = 1$ to n **do**

30: $Demanda(p_t) \leftarrow Demanda(p_t) - nrepPatron * NumPiezasAtendidas(p_b)$

31: **end for**

32: $i \leftarrow i + 1$

33: $listaSuperficies \leftarrow listaSuperficies + S_i$

34: $E \leftarrow E - PiezasAtendidas(S_i)$ {se decrementan las piezas atendidas}

35: **end while**

Comentando el algoritm, en el punto 5 se está realizando una copia de la lista de piezas WE ya que luego que se construye un patrón de corte no se debe actualizar las piezas en demanda. En el punto 14 se observa la condición de llenado de un contenedor la cual sigue un criterio GRASP para la elección de las piezas candidatas, así como para la cantidad de replicación. En el punto 28 se obtiene un número de replicación *nreppatron* para luego proceder en el punto 29 a la actualización de las demandas de piezas. En el punto 33 se agrega la solución S_i a la lista de superficies hasta ahora determinadas, procediéndose luego en el punto 34 a la actualización de las piezas atendidas.

4.2.4. ConstruccionGRASPFFDContenedor

Este procedimiento realiza la construcción de un patrón de corte, indicando las piezas que albergará la superficie S. Recibe como parámetros de entrada el número de superficie que se está analizando así como la lista de piezas en demanda.

Para su funcionamiento, se procede a realizar una copia de la lista de piezas $WE \leftarrow listapiezas$. Se entra en un bucle hasta que se termine la lista de piezas o se consiga llenar un contenedor. Se construye un RCL, se escoge una pieza aleatoria del RCL, se calcula la demanda que se puede atender de esa pieza, se encaja la pieza en el contenedor y se actualiza la lista de piezas WE.

El algoritmo implementado es como sigue:

1: $WE \leftarrow E$ {contiene una copia de la lista de piezas}
2: **while** $((AreaDisponible(S_nsup) > Min\{Area(p_r) : r \in WE\}) \wedge (WE \neq \emptyset))$ **do**
3: $minarea \leftarrow Min\{Area(p_r) : r \in WE\}$
4: $maxarea \leftarrow Min\{Max\{Area(p_r) : r \in WE\}, AreaDisponible(S_{nSup})\}$
5: $RCL \leftarrow \{r \in E : maxarea - \alpha(maxarea - minarea) \leq Area(p_r) \leq maxarea\}$
6: $u \leftarrow Min\{\lfloor \frac{AreaDisponible(S_{nSup})}{Area(p_k)} \rfloor, Demanda(p_k)\}$ {cantidad de piezas que se pueden atender}
7: $umax \leftarrow u$
8: $umin \leftarrow umax - \beta * umax$
9: $k \leftarrow Random(RCL)$
10: $cantidadreplica \leftarrow Random([umin, umax])$
11: $EncajarPieza(p_k, cantidadReplica, RCL)$ {Se encaja la pieza}
12: $WE \leftarrow WE - p_k$
13: **end while**

Comentando el algoritmo, en el punto 6 se calcula la demanda de piezas p_k que se pueden atender tomando un mínimo entre la cantidad de piezas que se pueden alojar en ese momento en el contenedor con la demanda disponible de la pieza p_k. En el punto 10 se halla la demanda de piezas que se van a encajar en el contenedor mediante un criterio GRASP, utilizando el parámetro de relajación β.

4.2.5. Simulación Dinámica

Realiza la compactación basada en dinámica de las piezas que se encuentran dentro del contenedor S_i.

Para su funcionamiento, se procede a realizar una configuración de las piezas que se encuentran en el contenedor, asociándoles elementos dinámicos tales como la masa, momentos de inercia, etc. Las piezas poseen una configuración inicial, definidas por su posición, orientación, y posiblemente con ciertas velocidades. Estas piezas inmersas en el campo gravitacional, actúan en dirección hacia el fondo del contenedor S_i. Las piezas adquieren comportamientos de movimiento e interacción con otras piezas y las paredes del contenedor. Los movimientos de las piezas son generados por la presencia de la gravedad, conforme evoluciona la dinámica de los cuerpos, se van considerando contactos y colisiones, cada cuerpo adquiere una cierta velocidad y aceleración; con esto sus estados (posición y orientación) varían constantemente. Esta evolución tiene como límite la situación casi estática de los cuerpos (los cuerpos no se mueven mucho), considerado como una situación de compactación [Ri01] En la Figura 4.1 podemos observar una situación de compactación.

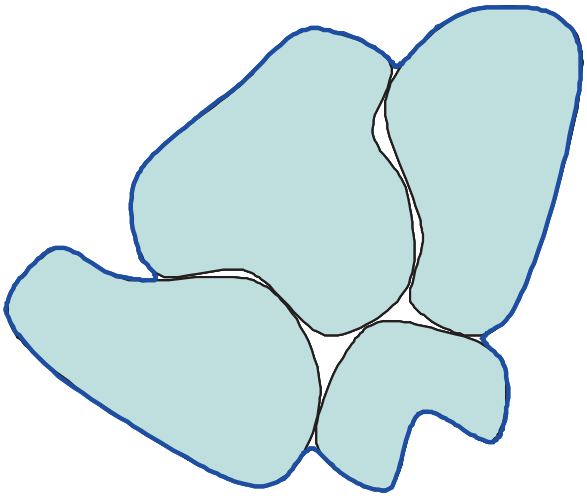


Figura 4.1: Situacion de Compactación.

4.2.6. RealizarAjusteSacudida

Para su implmentación, se procede a realizar un reacomodo de las piezas compactadas en un contenedor S_i tentando mejorar la configuración final de las mismas,

se aplican fuerzas a diferentes lados del recipiente tratando de realizar simular una sacudida.

4.2.7. CalcularAreasFinales

Para su funcionamiento, se procede a realizar el cálculo del área ocupada por las piezas de un patrón de corte S_i , hallando el porcentaje de utilización obtenido de restar el área del contenedor menos el área de las piezas consideradas en el contenedor S_i. Se realiza este cálculo hallando el cerco convexo que envuelve a las piezas alojadas en el contenedor S_i, como podemos observar en la Figura 4.2.

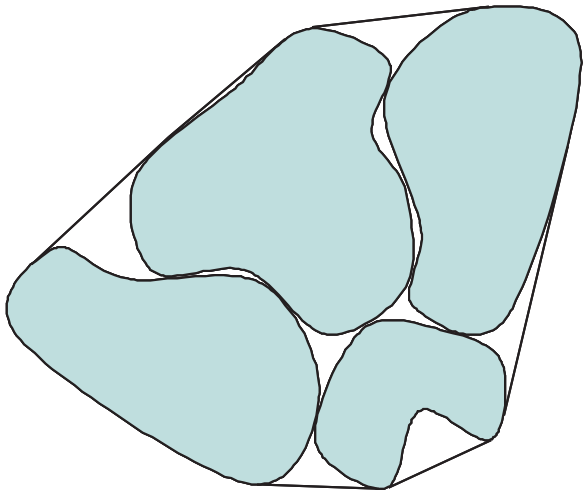


Figura 4.2: Cerco Convexo Resultante.

4.2.8. EncajarPieza

Para su funcionamiento, se procede a realizar el encaje de una cantidad determinada de piezas (*cantidadreplica*) al contenedor. Recibe como parámetro la pieza p_k a posicionar así como un contenedor S_i

El siguiente algoritmo implementa la función descrita.

1: $i \leftarrow 0$
2: **while** $i < cantidadreplica$ **do**
3: $InsertarPieza(S_i, p_k)$
4: $i \leftarrow i + 1$

5: **end while**

Comentando, en el punto 3 podemos observar al procedimiento $InsertarPieza$ el cual realiza el encaje de una pieza p_k al contenedor S_i.

4.2.9. ActualizarSolucion

Para su funcionamiento, se procede a realizar una comparación entre el número de superficies usadas por la presente iteración $superficiesusadas$ con el menor número de superficies encontradas hasta el momento $menornumreosuperficies$.

Se implementa con el siguiente algoritmo:

1: **if** $superficiesusadas < numerosuperficies$ **then**
2: $mejorsolucion \leftarrow listasuperficies$
3: $menornumerosuperficies \leftarrow superficiesusadas$
4: $i \leftarrow i + 1$
5: **end if**

Comentando el algoritmo, si es que se cumple la condición mostrada en el punto 1 se procede a la actualización de la solución.

Capítulo 5

Posicionamiento de Objetos

Luego de obtener el conjunto de piezas que conformarán un patrón de corte, se requiere encontrar una adecuada configuración inicial de las mismas, para lo cual se pretende buscar el mejor posicionamiento de estas piezas dentro de la superficie contenedora que los albergará.

Posicionar un conjunto de piezas dentro de una superficie contenedora es un problema dificil de resolver "NP-Hard", ya que los objetos irregulares que estamos tratando son de naturaleza no poligonal y se encuentran definidos por segmentos curvos; si lo objetos fueran de naturaleza poligonal, se podría usar como parámetros de posicionamiento, vértices, ángulos y lados que conforman los objetos, siendo esto un problema combinacional que ya ha sido resuelto en la literatura de varios formas.

En vista de esto, se tendrá que realizar un posicionamiento por aproximación, en el cual estos objetos no poligonales se convertirán en poligonales; para ello se tendrá que utilizar alguna técnica especializada de la geometría computacional tipo el cálculo del cerco convexo de cada objeto [FC91] o encontrar la caja orientada asociada a cada objeto [RC01].

En la literatura han sido estudiadas varias técnicas para realizar posicionamiento de objetos irregulares de constitucion poligonal dentro de superficies contenedoras. Destacan la técnica de No-Fit Polygon estudiado por Albano and Sapuppo [AS80], Dowsland and Dowsland [DD95], [DD02], Gomes and Oliveira [GO02]; la técnica de bottom left estudiada por Jakobs [Ja96]; la técnica de bottom left fill estudiada por Blazewicz et al [BHW93], Hopper [Ho+00].

Todas estas técnicas anteriormente mencionadas tratan objetos irregulares de constitución poligonal, por lo cual pretendemos realizar una técnica de posicionamiento que consiga tratar objetos irregulares no poligonales y así poder realizar un posicionamiento adecuado consiguiendo una configuración inicial de las piezas dentro de la superficie contenedora.

La organización del presente capítulo es de la siguiente manera: En la Sección 1 presentamos las técnicas de posicionamientos de objetos poligonales que seran de inspiración a nuestro propósito, en la Sección 2 mostraremos la técnica bottom left y por último en la Sección 3 presentaremos la técnica de posicionamiento vertical. Estas dos técnicas pasarán por una transformación para poder tratar objetos irregulares no poligonales.

5.1. Técnicas de posicionamiento para objetos poligonales

El problema de posicionar un conjunto de piezas irregulares ha venido estudiándose desde hace varios años atrás. Se han realizado estudios donde los objetos irregulares que se requiere tratar han sido aproximados usando alguna técnica, estas van desde simples formas regulares hasta distintos tipos de formas poligonales. Tanto es así que la forma más popular de aproximar un objeto irregular ha sido la utilización de rectángulos. Estos objetos han sido aproximados en simples formas regulares, las cuales son luego empacadas o posicionadas en el area disponible de la superficie que se está tratando.

Freeman and Shapira [FS75] hicieron el intento de incluir objetos irregulares de constitución poligonal en un polígono convexo, para luego encontrar el rectángulo requerido, iterativamente, basando este rectángulo en cada uno de los bordes del polígono. Otra alternativa de empacamiento rectangular que ha sido usada es aproximar todos los objetos irregulares de constitución poligonal dentro de polígonos idénticos, los cuales pueden ser usados para posicionarlos en la superficie. Estos polígonos pueden ser triángulos, cuadriláteros, pentágonos y hexágonos. Dori and Bean-Bassat [DB84] realizaron un algoritmo de empacamiento usando hexagonos, su estudio fue limitado solo a polígonos convexos. El algoritmo que propusieron consiste en tomar

los objetos irregulares y aproximarlos mediante un poligono, el cual minimice el área que circunscribe, luego de esto las piezas serán rodeadas mediante un hexágono que se irá posicionando en la superficie.

Algunas de las estrategias de posicionamiento estudiadas en la literatura han implicado que las piezas a colocar en una superficie sean ordenadas previamente, para luego ser posicionadas de acuerdo a una política de posicionamiento dado. El uso de algún método inteligente puede garantizar encontrar el mejor conjunto de piezas a posicionar; para esto, se necesita que el método escogido realize iterativamente diversos ordenamientos, los cuales generarán diversas colocaciones de las piezas dentro de la superficie. El método entregará luego de su ejecución las piezas a ser posicionadas en la superficie.

Conseguido el conjunto de piezas a posicionar, se utilizará alguna política de colocacion de piezas para poder determinar la ubicación de cada pieza dentro de la superficie conetendora. Para realizar esto tenemos, en la literatura, métodos como "bottom left", y otros enfoques los cuales se han venido haciendo populares recientemente por la incorporación de técnicas metaheurísticas como simulated annealing, tabu search, algoritmos genéticos.

La técnica de Bottom Left ha sido un enfoque muy popular en el cual luego de realizado el ordenamiento de las piezas a posicionar, cada pieza es movida tan lejos como sea posible al fondo de la superficie contenedora y luego tan lejos como sea posible tomando el lado izquierdo [Ja96]. La posición válida para cada pieza es encontrada cuando esta choca con la superficie en su lado inferior izquierdo. Asímismo, se puede considerar diferentes orientaciones para posicionar las piezas. La Figura 5.1 muestra una secuencia de posicionamiento de piezas irregulares de constitución poligonal, donde la secuencia de posicionamiento de las piezas es 1,2,3,4,5.

Este algoritmo ha sido estudiado muy estudiado para el caso de empacamientos regulares por Brown [Br80], Baker et al [BCR80], así como por Liu and Teng [LT99]. En el caso de empacamiento de piezas irregulares, el algoritmo ha sido estudiado por Oliveira et al [OF00], Amarral et al [ABJ90], Downsland et al [DD02]. Destacan los trabajos de Dowsland and Dowsland [DD95] donde se usa un algoritmo bottom left con un ordenamiento aleatorio de piezas.

La mayor desventaja de la técnica bottom left consiste en la creación de áreas

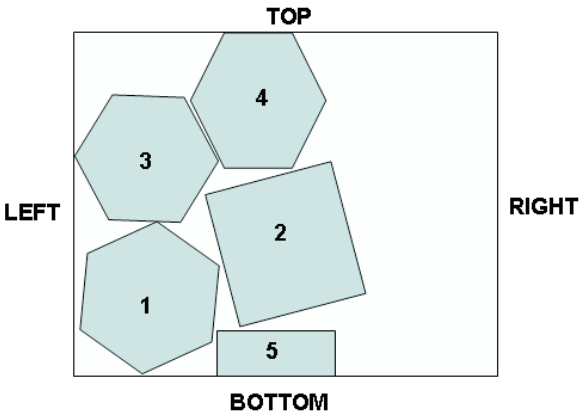


Figura 5.1: Funcionamiento Algoritmo Bottom Left.

vacías en la superficie, por tal motivo se desarrollo el algoritmo Bottom Left Fill el cual permite posicionar cada pieza irregular hacia la más baja posición disponible para tal objeto [Ho+00]. Este algoritmo es capaz de llenar los espacios vacíos que el algoritmo Bottom Left tradicional dejaba, ya que presenta una serie de movimientos Bottom Left lo cual permite la ubicación de una pieza en la más baja región de la superficie que se esta tratando.

La Figura 5.2 muestra el funcionamiento del algoritmo Botton Left Fill para una secuencia de piezas 1,2,3,4,5 donde se requiere posicionar la pieza 5. Tradicionalmente el algoritmo Bottom Left posicionaría la pieza 5 como se muestra en el caso (a), el algoritmo Bottom Left Fill posicionaría la pieza 5 cubriendo áreas vacías como se muestra en el caso (b).

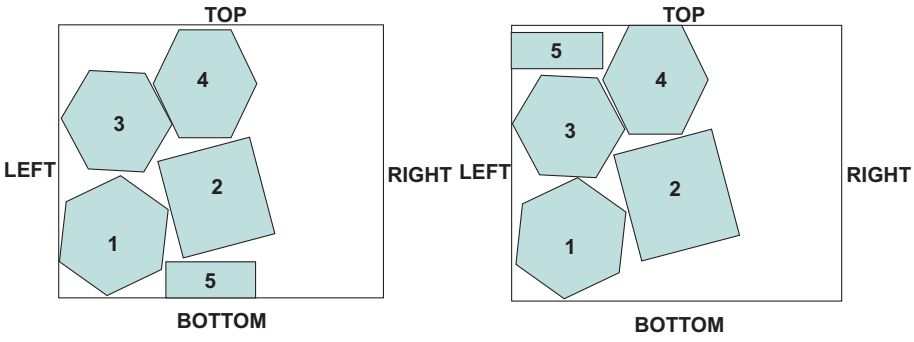


Figura 5.2: Funcionamiento Algoritmo Bottom Left Fill.

Algunos enfoques para resolver este problema han alcanzado buenos resultados, la mayoria de ellos han utilizado la técnica de “No-Fit polygon”la cual esta basada en generar una gran cantidad de posiciones probables para colocar una pieza irregular

en una superficie [GO02]. La Figura 5.3 muestra la construcción del No-Fit Polygon para dos polígonos P_1, P_2. El No-Fit polygon ha sido una poderosa técnica geométrica, pero existen una serie de casos que la hacen limitada para aplicaciones industriales. Puede fallar en formas que presentan agujeros, concavidades, en encajes perfectos como mosaicos, etc [BHK+05].

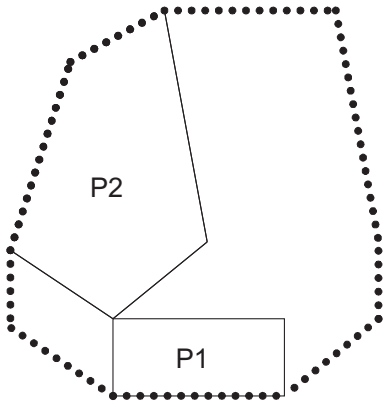


Figura 5.3: Funcionamiento Técnica No-Fit Polygon.

Asímismo se ha estudiado una técnica de posicionamiento vertical de las piezas respecto al centro de gravedad del contenedor que las alojará, donde las piezas serán posicionadas una detras de otra haciendo una hilera con respecto al centro del contenedor. Este método será descrito con mas detalle en la siguiente sección.

Trabajos anteriores en esta área son ampliamente específicos a un dominio particular de problema como por ejemplo tratar objetos irregulares que usualmente presentan tamaños y formas similares. Nosotros proponemos una técnica bottom left que implemente un algoritmo que trabaje con objetos no poligonales. Para implementar tal técnica necesitamos que los objetos sean de constitucion poligonal, por lo cual realizaremos una transformada a estos objetos irregulares no poligonales, calculando el cerco convexo de cada uno, luego del cual tendremos una representacion poligonal para cada objeto que estamos tratando de posicionar en la superficie.

5.2. Técnica Bottom Left

Luego de realizado el proceso de transformación de piezas irregulares no poligonales a poligonales realizada por el cálculo del cerco convexo de cada pieza obtenemos

un conjunto de polígonos N los cuales se tendrán que posicionar en la superficie S siguiendo el criterio de posicionamiento Bottom Left que presentamos a continuación.

El algoritmo de posicionamiento bottom left que presentamos va tomando cada polígono a posicionar de una lista N y lo posiciona en la parte inferior izquierda de la superficie contendera S donde exista espacio disponible. Para realizar un adecuado posicionamiento del polígono en la superficie se deberá de ordenar previamente el conjunto de vértices que conforman la superficie contenedora así como tambien se requiere ordenar el conjunto de vertices que conforman el polígono que se esta posicionando (Los vertices se ordenan en criterio Bottom Left). Una vez ordenados los vértices tanto del contenedor como del polígono, se obtienen los puntos de contacto de la superficie $ptoSuperficie$ y del polígono $ptoPoligono$ y se procede a posicionar el polígono en la superficie contenedora. La Figura 5.4 muestra el resultado de posicionamiento del polígono A en la superficie contenedora S.

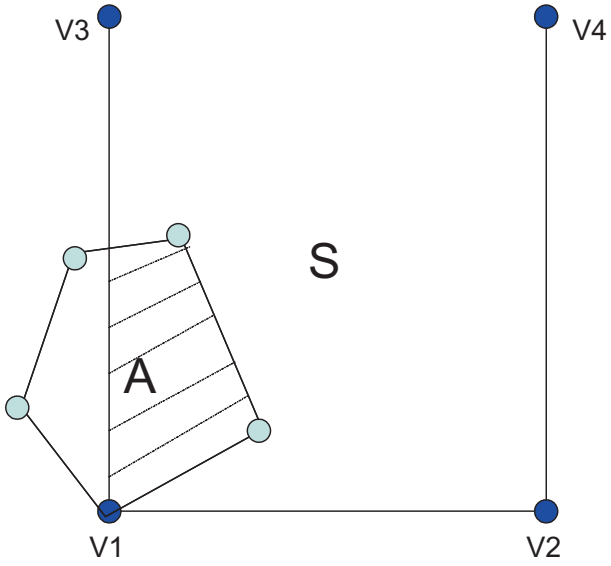


Figura 5.4: Posicionamiento de Polígono en Superficie.

Como podemos observar en la Figura 5.4 al posicionar el polígono A existe intersección con la superficie, por lo cual debemos de hacer una traslación del polígono posicionado a una nueva posición, tomando en cuenta el espacio que existe entre el vértice más a la izquierda que se encuentra fuera de la superficie con el punto de intersección. A esta distancia la llamamos xd, para poder hallar el nuevo punto de posicionamiento deberemos de analizar si es factible el traslado del polígono hasta esta nueva posicion, por lo cual debemos calcular el espacio posible para traslado xs. si este espacio es mayor o igual a la distancia de traslado que hemos determinado

procedemos a posicionar el polígono A en esta nueva ubicación. La Figura 5.5 muestra la determinación del nuevo punto de traslado para el polígono A en la superficie contenedora S

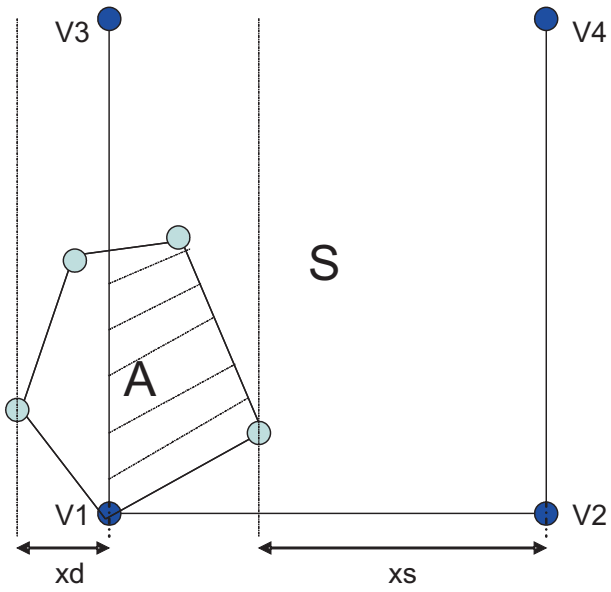


Figura 5.5: Determinación de Punto de Traslado del polígono en la superficie.

Ubicada la posición del polígono A dentro de la superficie contenedora S libre de intersecciones se procede a realizar el cálculo del nuevo centro del polígono $Nc = Ca - P_1$, para luego determinar los nuevos vértices que tendra la superficie contenedora a consecuencia del posicionamiento del polígono; estos se calculan de acuerdo a los puntos de contacto que se originan entre la superficie S y la superficie rectangular que encierra al poligono posicionado, estos vértices se agregan a la lista de vértices que conforman la superficies contenedora S. La Figura 5.6 muestra el cálculo del nuevo centro del polígono asi como la determinacion de los nuevos vértices generados por dicho polígono.

Luego que hemos actualizado la lista de vértices de la superficie contenedora S procedemos a decrementar el área disponible para posicionamiento en el valor del área que determina el polígono A, esta variable *areaDisponible* servirá para que el algoritmo controle hasta donde será posible posicionar polígonos en la superficie contenedora S. A continuación tomamos el siguiente polígono de la lista N y repetimos iterativamente el proceso hasta que se esten posicionados todos los polígonos dentro de la superficie S o ya no exista área disponible para posicionar polígonos. La figura 5.7 muestra la superficie contenedora S luego de posicionar un conjunto de polígonos.

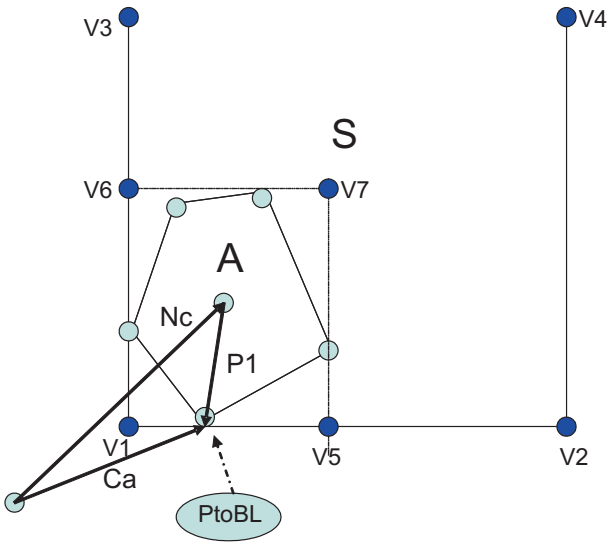


Figura 5.6: Cálculo de Nuevo Centro del Polígono y Determinación de Nuevos Vértices

A continuación presentaremos el pseudocódigo que refleja el funcionamiento del algoritmo bottom left propuesto.

5.2.1. Pseudocódigo Técnica Bottom Left

Sea $N = \{p_1, p_2, ...p_n\}$ el conjunto de polígonos a posicionar dentro de la superficie S, Nc_k $\forall k \varepsilon N$ el vector que representa el nuevo centro del polígono p_k; Ca_k el vector formado por un vértice del polígono p_k con el origen de coordenadas, i es el iésimo vértice del polígono p_k.

1: $k \leftarrow 1$
2: $areaDisponible \leftarrow Area(S)$
3: **while** $((k \leq n) \wedge (AreaDisponible > 0))$ **do**
4: $verticesSuperficie \leftarrow OrdenarVerticesBL(S)$
5: $verticesPoligono \leftarrow OrdenarVerticesBL(p_k)$
6: $ptoSuperficie \leftarrow ObtenerVerticeBL(S, verticesSuperficie)$
7: $ptoPoligono \leftarrow ObtenerVerticeBL(p_k, verticesPoligono)$
8: $PosicionarBL(S, p_k, ptoSuperficie, ptoPoligono)$
9: **while** $(Overlap(p_k, S) = true)$ **do**
10: **if** $(espacio \geq distancia)$ **then**
11: $ptoSuperficie \leftarrow ObtenerVerticeBL(S, verticesSuperficie)$
12: $ptoSuperficie.x \leftarrow ptoSuperficie.x + distancia$

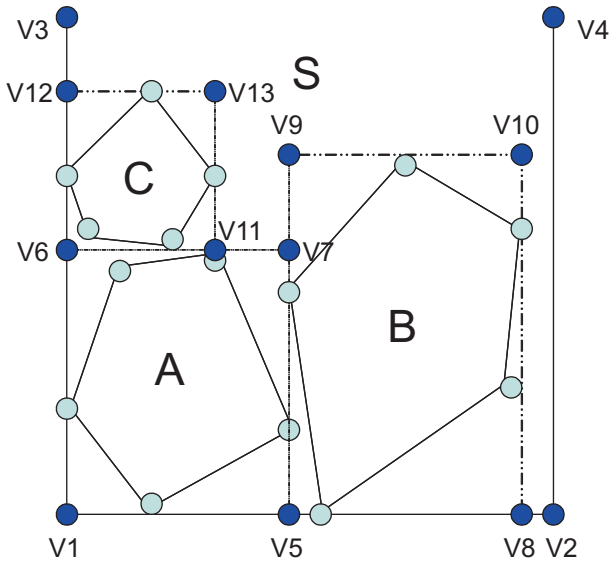


Figura 5.7: Posicionamiento de Polígonos en Superficie Contenedora.

13: $PosicionarBL(S, p_k, ptoSuperficie, ptoPoligono)$
14: **else**
15: $ptoSuperficie \leftarrow ObtenerVerticeBL(S, verticesSuperficie)$
16: $PosicionarBL(S, p_k, ptoSuperficie, ptoPoligono)$
17: **end if**
18: **end while**
19: **if** $(Overlap(p_k, S) = false)$ **then**
20: $AgregarVerticesSuperficie(S, p_k, ptoSuperficie)$
21: $S \leftarrow S - p_k$
22: $areaDisponible \leftarrow areaDisponible - Area(S)$
23: $k \leftarrow k + 1$
24: **end if**
25: **end while**

5.2.2. Comentario

En el punto 4 y 5, el algoritmo ordena los vértices de la superficie S y el polígono que se esta tratando de acuerdo al criterio Bottom Left, donde primeramente se hace el ordenamiento de las ordenadas de los vértices para luego realizar el ordenamiento sobre las absisas. En el punto 9 el algoritmo verifica si existe alguna intersección a consecuencia del posicionamiento del polígono en la superficie contenedora S; en el

punto 10 el algoritmo verifica si existe intersección, luego en el punto 11 se procede a trasladar la pieza a un nuevo punto obtiéndose el vértice más a la izquierda para luego en el punto 12 determinar el nuevo punto de traslado del polígono. En el punto 20 el algoritmo agrega los nuevos vértices generados por el posicionamiento del poligono en la superficie a la lista de vertices de la superficie S.

La Figura 5.8 muestra el posicionamiento de una superficie contenedora con sus respectivas piezas posicionadas luego de aplicar la técnica bottom left.

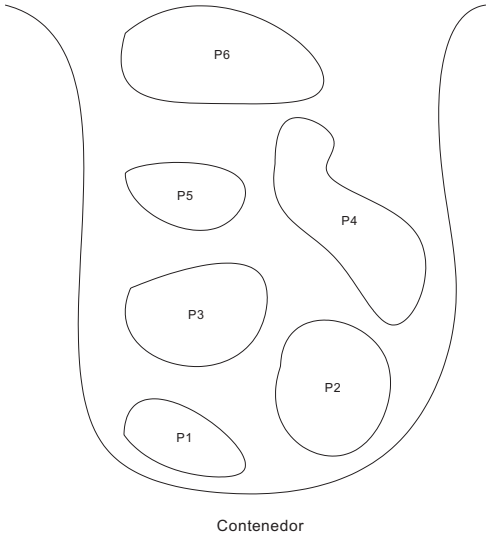


Figura 5.8: Ejemplo Posicionamiento Bottom Left

5.3. Técnica de Caída Vertical

Esta técnica pretende realizar el posicionamiento de las piezas una tras otra en forma vertical tomando como punto de referencia el centro de gravedad de la superficie contenedora. El conjunto de piezas a posicionar deberá colocarse una tras otra formando una hilera vertical, tendiendo en consideración una distancia de separación entre cada objeto a colocar.

Sea $N = \{p_1, p_2, ...p_n\}$ el conjunto de piezas irregulares poligonales a posicionar dentro de la superficie S; el vector $cc = (cc[0], cc[1], cc[2])$ las coordenadas x, y, z del centro de gravedad del contenedor S; $cp_k = (cp_k[0], cp_k[1], cp_k[2])$ $\forall k \varepsilon N$ las coordenadas de x, y, z de cada pieza irregular y r_i el radio del círuclo concentrico que circunscribe a la pieza irregular p_i $\forall i \varepsilon N$.

Se buscará ir variando la coordenada y (respecto al centro de gravedad del contenedor) en una distancia dada por el cálculo de los radios de los círculos concéntricos originados entre el centro de una pieza ya posicionada con una pieza a posicionar. La coordenada x de cada pieza permanecerá fija y corresponderá a la coordenada x del centro de gravedad de la superficie contenedora.

Cada pieza irregular irá variando su coordenada tomando como referencia la anterior pieza posicionada; por ejemplo si queremos posicionar la segunda pieza p_2, las coordenadas de esta pieza serían $cp_2 = cp_1 + ((r_1 + r_2), 0, 0)$, generalizando si queremos posicionar la pieza p_k sus coordenadas serían $cp_k = cp_{(k-1)} + ((r_{(k-1)} + r_k), 0, 0)$. La Figura 5.9 muestra el posicionamiento en forma vertical para un contenedor con sus respectivas piezas irregulares.

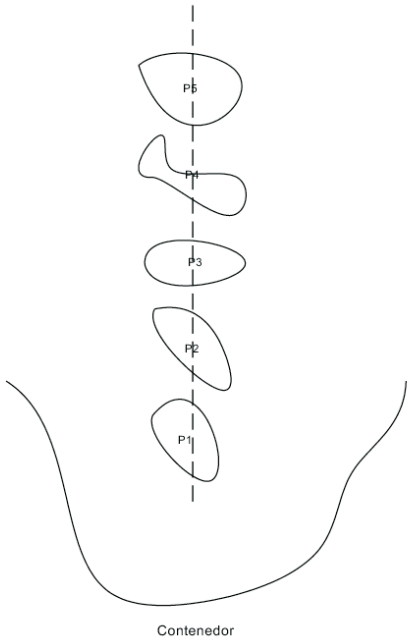


Figura 5.9: Ejemplo Posicionamiento Vertical

Esta técnica de posicionamiento surge bajo el criterio de que un cuerpo en reposo puede ser soltado bajo acción de la gravedad y realizará una caída libre con dirección hacia un fondo. Como consecuencia utilizamos como fondo a la superficie contenedora y como cuerpos en reposo a las piezas irregulares las cuales luego de posicionadas serán soltadas.

Capítulo 6

Dinámica de los Cuerpos

6.1. Introducción

Realizar la simulación del movimiento de un cuerpo en el universo, demanda un análisis de una serie de casos; como análisis de contactos e intersecciones (interferencias).

Cuando un cuerpo se mueve por traslación y rotación, los valores de su estado varían constantemente. El análisis de ese estado es hecho en cada intervalo de tiempo, el cual es llamado paso de animación. Es aquí donde se analiza si el cuerpo presenta intersección, contacto o simplemente está alejado con respecto al otro. En el presente capítulo veremos como se realiza dicho análisis y se sale de una condición de interferencia o contacto.

6.2. Características dinámicas de los cuerpos

6.2.1. Características inerciales

Entre las caraterísticas inerciales tenemos a la inercia, masa y momento de inercia.

La propiedad de inercia de los cuerpos se aclara en la primera ley de Newton: Todo cuerpo conservará su estado de reposo o de movimiento rectilíneo y uniforme mientras fuerzas externas aplicadas a él no le hagan variar ese estado. En tal sentido,

cualquier cuerpo conservará su velocidad mientras no exista una fuerza que lo obligue a variarla.

La masa del cuerpo es la medida de la inercia del cuerpo en el movimiento de traslación. Se mide por la relación entre la magnitud de la fuerza aplicada y la aceleración provocada por ella:

$$m = \frac{\overline{F}}{\overline{a}};$$
$$[m] = M$$

donde m es la masa, F es la fuerza, a es la aceleración. La medición de la masa del cuerpo se basa en la segunda ley de Newton: La variación del movimiento es directamente proporcional a la fuerza actuante desde fuera y se produce en el sentido en que está aplicada esta fuerza.

En un cuerpo absolutamente rígido existen tres puntos, cuyas situaciones coinciden: el centro de masa, el centro de inercia y el centro de gravedad. Sin embargo, estos son conceptos completamente diferentes. En el centro de masa se interceptan las direcciones de las fuerzas, cada una de las cuales provoca el movimiento de traslación del cuerpo, mientras que el centro de inercia es el punto de aplicación de la resultante de todas las fuerzas ficticias de inercia y el centro de gravedad es el punto de aplicación de la resultante de todas las fuerzas de gravedad.

Se denomina momento de inercia a la medida de la inercia del cuerpo en el movimiento de rotación. El momento de inercia del cuerpo respecto al eje es igual a la suma de los productos de las masas de todos los puntos materiales del cuerpo por los cuadrados de sus distancias hasta el eje dado:

$$I = \sum(m_i r_i^2);$$
$$[I] = ML^2$$

6.2.2. Características de fuerza

En esta subsección describiremos a la fuerza, momento de una fuerza, impulso de una fuerza y momento del impulso de una fuerza.

La medición de la fuerza, de la misma forma que la de la masa, está basada en la segunda ley de Newton:

$$\overline{F} = m.\overline{a};$$
$$[F] = MLT^{-2}$$

Una fuerza, aplicada a un cuerpo dado, provoca su aceleración. La fuente de la fuerza es otro cuerpo; por consiguiente, interactúan dos cuerpos. De esta forma, existe la acción del segundo cuerpo sobre el primero, y la reacción del primer cuerpo aplicada sobre el segundo. Como la acción y la reacción están aplicadas a diferentes cuerpos, no se les puede componer, o sea, sustituir por una resultante. Según la tercera ley de Newton - Para cada acción siempre existe una reacción de igual magnitud, pero en sentido contrario - las acciones de dos cuerpos uno sobre el otro, siempre son iguales y opuestas por su sentido. Hay que entender claramente que esta ley es válida sólo para los sistemas inerciales de referencia

Se denomina momento de una fuerza a la medida de la acción de rotación de una fuerza sobre un cuerpo. Se determina por el producto del módulo de la fuerza por su brazo:

$$M_z = \overline{F}.d;$$
$$[M_z] = ML^2T^{-2}$$

Se denomina impulso de una fuerza a la medida de la acción de la fuerza sobre el cuerpo en un determinado intervalo de tiempo (en el movimiento de traslación). Al final del intervalo de tiempo es igual a la integral definida a partir del impulso elemental de fuerza, donde los límites de integración son los instantes de comienzo y final del intervalo de tiempo de acción de la fuerza:

$$\overline{S} = \int \overline{F}dt;$$
$$[S] = MLT^{-1}$$

Se denomina impulso del momento de una fuerza a la medida de la acción del momento de una fuerza respecto a un eje durante un intervalo dado de tiempo (en el

movimiento de rotación).

Al final del intervalo de tiempo es igual a la integral definida a partir del impulso elemental del momento de la fuerza; los límites de la integral son los instantes de comienzo y final del intervalo de tiempo dado:

$$\overline{S}_z = \int M_z \overline{F} dt;$$
$$[S_z] = ML^2T^{-1}$$

Como resultado del impulso, tanto de la fuerza como del momento de la fuerza, se originan variaciones de los movimientos, que dependen de las propiedades inerciales del cuerpo y que se ponen de manifiesto en la variación de la velocidad (cantidad de movimiento, momento angular).

La cantidad de movimiento es la medida del movimiento de traslación del cuerpo, que caracteriza su capacidad de trasmitirse a otro cuerpo en forma de movimiento mecánico. Se mide por el producto de la masa del cuerpo por su velocidad:

$$\overline{P} = m.\overline{v}$$

En mecánica newtoniana, el momento angular de una masa puntual, es igual al producto vectorial del vector de posición $\overline{r}$ (brazo), del objeto en relación a la recta considerada como eje de rotación, por la cantidad de movimiento (también llamado momento lineal o momento). Frecuentemente se lo designa con el símbolo $\overline{L}$:

$$\overline{L} = \overline{r} \times \overline{p}$$

$$\overline{L} = \overline{r} \times m.\overline{v}$$

6.3. Generación de movimiento

En un sistema de animación, existe un módulo de generación de movimiento el cual es ejecutado en cada intervalo de tiempo (paso de tiempo). Este módulo esta basado en la dinámica de Newton-Euler, la cual relaciona el movimiento con las fuerzas y torques aplicados sobre los cuerpos, a través de unas ecuaciones diferenciales

ordinarias (EDO). Estas ecuaciones son integradas numéricamente para asi poder obtener los estados de los cuerpos compuestos por sus posiciones, orientaciones y velocidades; con ello se analizan las diversas situaciones de interferencia que se pueden generar al momento que los cuerpos se encuentran en movimiento, asi como detectar también si existen situaciones de contacto.

6.3.1. Estado de un cuerpo en movimiento

Un cuerpo en un instante t, tiene un estado $Y(t)$ el cual esta compuesto por la posicion $x(t)$ de su centro de masa en el espacio, la orientacion de su sistema de coordenada local $R(t)$, velocidad linear $v(t)$ de su centro de masa, y su velocidad angular $w(t)$. El estado del cuerpo lo podemos ver como:

$$Y(t) = \begin{pmatrix} x(t) \\ R(t) \\ v(t) \\ w(t) \end{pmatrix}. \tag{6.1}$$

Se aconseja que cuando los cuerpos se encuentra en movimiento dinámico, el elemento de estado $v(t)$ sea sustituido por el momento lineal $P(t)$, y $w(t)$ por el momento angular $L(t)$. Con las observaciones mencionadas podemos denotar el estado de un cuerpo en movimiento dinámico como:

$$Y(t) = \begin{pmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{pmatrix}. \tag{6.2}$$

Como se expuso anteriormente el movimiento de un cuerpo en el tiempo es determinado por una ecuacion diferencial basada en la formulación dinámica de Newton-Euler:

$$\frac{d}{dt}Y(t) = \frac{d}{dt}\begin{pmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{pmatrix}. \tag{6.3}$$

$$\frac{d}{dt}Y(t) = \frac{d}{dt}\begin{pmatrix} v(t) \\ w(t) \times R(t) \\ F(t) \\ T(t) \end{pmatrix}. \tag{6.4}$$

donde $F(t)$ y $T(t)$ son la fuerza de torque resultante actuando sobre el cuerpo.

Podemos relacionar las variables de la ecuacion de la siguiente manera [Ri96]:

$$\begin{aligned} v(t) &= P(t)/m, \\ F(t) &= \tfrac{ddt}{P}(t), \\ T(t) &= \tfrac{ddt}{L}(t), \end{aligned} \tag{6.5}$$

donde m es la masa del cuerpo,

El momento angular total $L(t)$ de un cuerpo rígido esta relacionado con la inercia $I(t)$ de un cuerpo a través de la ecuacion:

$$L(t) = I(t).w(t) \tag{6.6}$$

La ecuacion diferencial puede ser integrada en cada paso de tiempo utilizando métodos numéricos para resolver ecuaciones diferenciales ordinarias (ODE); por ejemplo el método de Euler, de Runge-Kutta o algún otro método híbrido, dependiendo de la exactitud deseada. Para propósitos del presente trabajo se utilizó el método de Euler.

6.3.2. Análisis de la evolución del movimiento

Los estados de todos los cuerpos en un instante t forman el estado del sistema $Y(t)$; es aquí donde se registra la evolución de todos los objetos hasta el instante t.

Dado el estado inicial $Y(t_0)$, todo el sistema evoluciona para el próximo estado que es registrado en un paso de tiempo el cual llamamos t_{step}. Si aquí no existiera ninguna interferencia espacial entre los objetos, en el intervalo $[t_0, t_0 + t_{step}]$, el nuevo estado adquirido por el sistema es $Y(t_0 + t_{step})$, el cual pasa a ser el estado inicial para el próximo paso de integración.

Si se detecta alguna interferencia en $t_0 + t_{step}$, el estado inicial $Y(t_0)$ es recuperado. Es aqui donde se debera tomar un instante t_c de forma tal que $t_0 < t_c < t_0 + t_{step}$; se usara alguna técnica de convergencia de $t_{ii \in I_c}$ para que en instantes intermedios para t_c se obtengan los estados intermedios $Y(t_i)_{i \in I_c}$, los cuales sean situaciones próximas al contacto $Y(t_c)$. El mayor interes en este proceso es que I_c tenga el menor número de elementos posibles para una convergencia rápida en t_c. Una vez determinado el estado $Y(t_c)$, son tratadas sus interferencias respectivas, para luego completar la iteración de este último estado hasta el estado de contacto $t_0 + t_{step}$.

6.4. Verificación de no Interpenetración

Luego de haber detectado una interferencia en un espacio de tiempo, se procede a realizar un análisis de interferencias. En [Ri01] podemos encontrar tres niveles de detección de interferencias: alto nivel o aproximado, nivel de aproximación por jerarquías y niveles en detalle.

Para poder realizar el análisis de detección de interferencias se utiliza una estructura jerárquica que envuelve pedazos de contornos de objetos y sus detalles [RCV02]; esta estructura permite descartar rápidamente las partes de los objetos que no están en posible interferencia, y permite realizar una comparación rápida de intersecciones.

Las estructuras jerárquicas que se pueden usar son árboles binarias, cuaternarias u octales, y las envolturas pueden ser esferas, cajas rectangulares isotéticas (cajas de aristas paralelas al sistema de coordenadas del universo), elipses y cajas orientadas.

La envoltura deber ser la mas ajustada o adaptada posible al segmento que se va a envolver, esto incrementa la posibilidad de detectar los segmentos en interferencia. Como para el presente trabajo de tesis estamos utilizando objetos de contornos poligonales, utilizaremos un árbol de cajas orientadas, las cuales han mostrado en la literatura mayor rendimiento para el análisis de interferencias. La figura 6.1 muestra

un árbol de cajas orientadas:

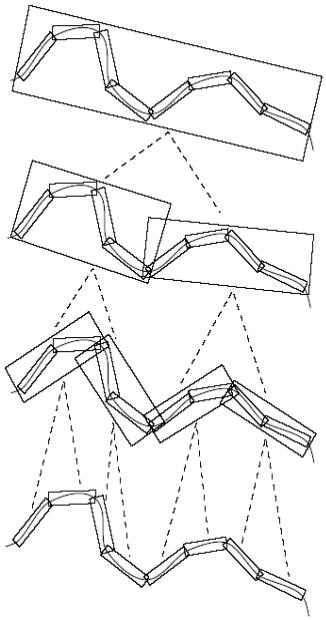


Figura 6.1: *Cajas Orientdas.*

EL método de árbol de cajas orientadas envolventes permite realizar una comparación rápida en tiempo constante intersectando las cajas que se encuentran en un mismo nivel de jerarquía, ya que existen dos cajas a comparar, se tiene un máximo de cuatro combinaciones en cada nivel del árbol, esta comparación se va realizando uno a uno, verificando si existe intersección. La verificación de interferencias es hecha recursivamente comenzando por la raíz de los árboles y yendo a través de los hijos. Al finalizar, en caso existan cajas básicas registradas, se procede a realizar analíticamente la intersección de los objetos. La figura a continuación muestra dos situaciones de objetos.

Cada par de cajas básicas en intersección puede incluir segmentos de superficies que están en tres situaciones: intersectados, en contacto o alejados.

Dos segmentos están intersectados cuando existe un mínimo punto de intersección entre dos segmentos, esto se da cuando existe una parte de un segmento en el interior del otro. Esta situación indica que se deberá de realizar una nueva iteración con un paso de tiempo menor, ya que en la presente iteración fue dado un paso que permitió que los objetos se interpenetren, por lo cual no será necesario continuar con la verificacion de otros pares de cajas en interferencia.

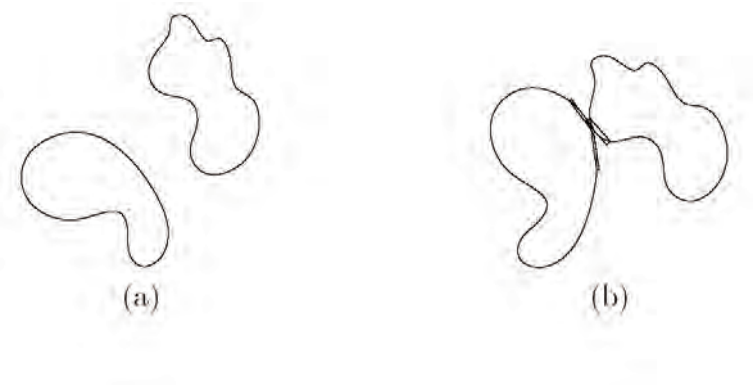


Figura 6.2: *Situación de dos objetos: (a) objetos separados; (b) objetos en contacto.*

Dos segmentos están en contacto cuando existe un punto sobre cada segmento de forma que son geométricamente proximos, con una tolerancia λ permitida. Estos son los puntos $f(t_f)$ e $g(t_g)$, dos segmentos f y g, respectivamente, estan suficientemente proximos cuando el vector $d(t_f, t_g) = f(t_f) - g(t_g)$ tiene la dirección de la normal comun a dos de los segmentos $0 \leq \| \ d(t_f, t_g) \ \| \leq \lambda$. Aqui solamente consideraremos cuerpos en contacto cuando exista intersección tangencial entre ambos cuerpos o cuando estos cuerpos esten sumamente próximos una tolerancia permitida.

Segmentos alejados son aquellos que no se intersectan y no estan en contacto. De esta manera, cuando ocurre una situación en contacto o alejados, el proceso de detección de puntos de contacto continua con los otros pares de cajas de interferencia.

La aproximación a los puntos de contacto cuando es detectada una interpenetracion es hecha mediante la tecnica de bisección. Esta técnica consiste en la partición binaria de cada paso de tiempo donde se disminuye el tamaño cuando existe una interpenetración y se incrementa la mitad de paso anterior cuando todos estan alejados.

Cuando ocurre una situación de contacto, cada par de puntos de contacto son almacenados en una lista de puntos de contacto. Estos puntos seran tratados para evitar interpenetraciones en iteraciones dinámicas futuras.

6.4.1. Análisis de Colisiones

Según [Ri01], el tratamiento de colisiones y contactos tiene como objetivo principal evitar las interpenetraciones entre objetos, para ello se modifica apropiadamente la tendencia de sus movimientos.

Las colisiones causan discontinuidad en las velocidades de los cuerpos; en cuanto los contactos continuos causan la discontinuidad en las aceleraciones debido a las fuerzas que actúan en cada cuerpo. Las discontinuidades de las velocidades es establecida por impulsos, de acuerdo con las leyes de conservación de momento lineal y angular de los cuerpos. Las discontinuidades de aceleraciones envuelven adición o eliminación de restricciones de aceleraciones de los objetos.

La secuencia normal para resolver colisiones y contactos consiste en primero tratar colisiones en el nivel de velocidades y después alterar sus aceleraciones, ya que el cálculo de las aceleraciones generalmente requieren que las alteraciones generadas por las variaciones de las velocidades sean conocidas.

Cuando existe una colisión todas las fuerzas externas son ignoradas, solamente son consideradas las fuerzas impulsivas. La fuerza impulsiva, llamada tambien fuerza de colisión, (f_c), actúa entre los cuerpos en contacto; esta es generada como una reacción de un cuerpo por la acción de otro en el punto de contacto. Esta fuerza es compuesta por una fuerza de restricción de interpenetración (f_n) en una fuerza de fricción (f_t).

$$f_c = f_n + f_t \tag{6.7}$$

La fuerza de restricción no produce trabajo, es una fuerza en dirección de la normal n, la fuerza de friccion f_t tiene dirección t y produce trabajo por disipación de energía cinética debido a la fricción de contacto. Para el presente trabajo de tesis hemos considerado una fricción mínima, casi 0.

La fuerza de contacto actuando sobre los puntos de contacto durante el intervalo de tiempo infinitesimal, $\triangle t$ es el impulso definido como:

$$J = \int_{\triangle t} f_c dt \tag{6.8}$$

El impulso J es el parámetro vectorial que hace mudanza de las velocidades de los cuerpos para evitar la interpenetración. La conservación de momento lineal y angular establece la relación del impulso con las velocidades de los dos cuerpos en colisión. Siendo los cuerpos A y B en colisión se tiene:

$$
\begin{aligned}
m_a v_a^+ &= m_a v_a + J \\
m_b v_b^+ &= m_b v_b + J \\
I_a w_a^+ &= I_a w_a + r_1 x J \\
I_b w_b^+ &= I_b w_b + r_2 x J
\end{aligned}
\tag{6.9}
$$

I_i es el momento de inercia del cuerpo i, v_i y w_i son velocidades lineal y angular antes de la colisión, v_i^+ y w_i^+ son las velocidades lineal y angular de i después de la colisión. En realidad los elementos que permanecen constantes durante la colisión son I_i, la masa m_i y la posición de contacto r_i.

Para poder resolver estas ecuaciones de conservación de momentos, es necesario considerar las condiciones de contacto: restitución de colisión y fricción de contacto. En [Ri01] podemos encontrar el detalle de la resolución de estas ecuaciones, así como un análisis detallado de las condiciones de contacto para aplicar impulsos.

La figura a continuación 6.3 muestra dos cuerpos en contacto, así como la forma en que se realiza el análisis de velocidades lineales y angulares para aplicar impulso.

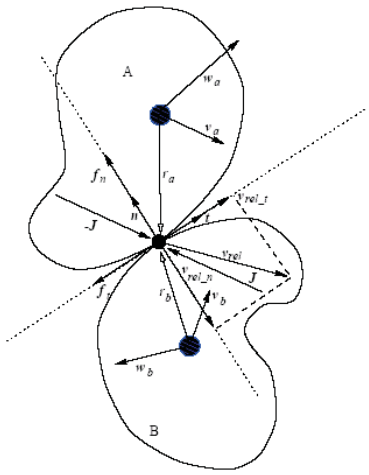


Figura 6.3: *Objetos en Contacto donde se aplicara impulso.*

6.5. Situación Estática

Esta situación se da cuando los cuerpos están en microcolisiones, es decir cuando las velocidades lineales y angulares son muy pequeñas, relativamente cercanas a cero;

con lo cual se puede decir que los cuerpos en este momento se encuentran en reposo. A esta condición llamamos contacto continuo.

Se toma como hipótesis de que cada par de objetos que se encuentren en contacto continuo están en constantes vibraciones mínimas, siguiendo una trayectoria balística [Ri01]. La condición de contacto continuo se da para un punto p cuando la componente normal de su velocidad relativa es considerada cero $v_{rel-n} \simeq 0$.

Capítulo 7

Análisis de Compactación

Cuando los objetos (piezas y superficies irregulares) se encuentran en una situación compactada (casi estática) es requerido realizar un análisis para poder medir el área del sector ocupado por estos objetos dentro de las superficies que los albergan.

Se entiende por área del sector ocupado, a la unión del área de los objetos (piezas irregulares) con el desperdicio (pedazos de áreas sobrantes). Es así que surge como necesidad conocer esta área para poder determinar exactamente el grado de ocupación de los objetos dentro de sus respectivas superficies contenedoras.

Consideremos como parámetro para estimar el área total de desperdicio A_d, la cual resulta de restar el área del sector ocupado A_s con el área total de los objetos A_o; esto es:

$$A_d = A_s - A_o \tag{7.1}$$

El área total de los objetos es simple de calcular, se realiza sumando el área de cada objeto colocado dentro del sector ocupado. El área difícil de calcular es A_s, la cual demandará el uso de técnicas avanzadas de computación gráfica.

Para poder determinar el área del sector ocupado utilizaremos la teoría de cerco convexo, mediante la cual se encontrará el menor conjunto convexo que contenga los puntos que determinen los objetos compactados.

En la literatura se han encontrado diversas técnicas para calcular el cerco con-

vexo en un conjunto de puntos en el plano; en el presente trabajo de tesis hemos implementado un algoritmo de QuickHull que presenta complejidad media de $n*\log n$. Este algoritmo determinará un conjunto de vértices que conformen el cerco convexo requerido.

El presente capitulo se encuentra distribuido de la siguiente manera: en la sección 1 se presenta la definición del problema de cerco convexo; en la sección 2 se presenta la formulación del cerco convexo a estudiar; en la sección 3 se describen diversos algoritmos para el problema de cerco convexo y en la sección 4 se describe el algoritmo de QuikHull a implementar para encontrar el cerco convexo de los objetos compactados en la superficie contenedora.

7.1. Definición del Problema de Cerco Convexo

El cerco convexo de un conjunto de puntos de dos dimensiones es definido como el menor polígono convexo que contiene todos los pontos en el plano. Cada punto del conjunto determinado es un vértice de ese polígono o se encuentra en su interior.

Figueiredo [FC91] define el problema del cerco convexo a partir de un conjunto finito de pontos C, donde determinar el cerco significa calcular el menor conjunto convexo que contiene tales puntos.

Si tenemos un conjunto $C = \{p_1, p_2, ..., p_n\}$ de puntos en el plano, obtener el cerco convexo de C implica determinar cuales de los puntos de C son vértices del cerco convexo de C (que será denotado por $conv(C)$ que es el conjunto de todas las combinaciones convexas de elementos de C) y ordenar esos puntos circularmente, de acuerdo con su ocurrencia en la frontera de $conv(C)$, [FC91].

La Figura 7.1 que presentamos a continuación muestra el cerco convexo de un conjunto de 27 pontos en el plano.

7.2. Area del sector ocupado

El sector ocupado puede ser calculado en forma aproximada, pues no existe un parámetro para saber si un segmento debe ser considerado o no como desperdicio,

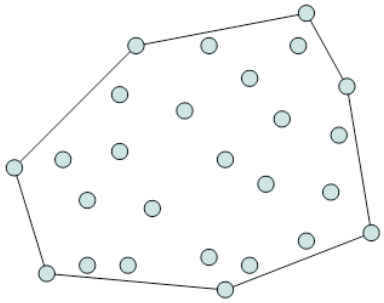


Figura 7.1: Cerco convexo a partir de 27 puntos.

en términos de patrón de corte con objetos irregulares. Como se muestra en la Figura 7.2, un sector ocupado puede ser considerado un cerco convexo adaptado (FCA) envolviendo ajustadamente los objetos compactados (7.2 (a)), un cerco convexo poligonal (FCP) envolviendo ajustadamente los objetos compactados (7.2(b)), y un cerco adaptado (FA)envolviendo ajustadamente a los objetos compactados (7.2 (c)).

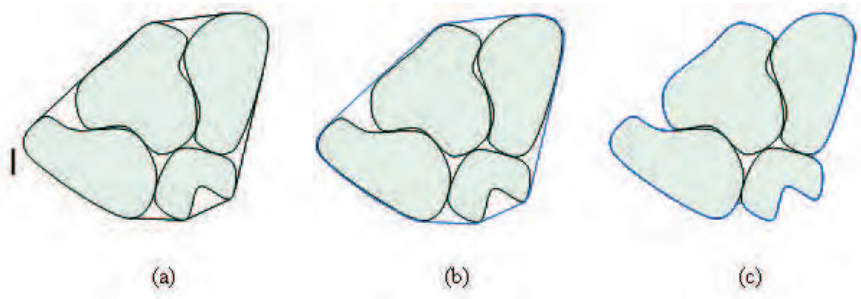


Figura 7.2: Area del sector ocupado.

Dada una situación de los objetos compactados, considerando las tres configuraciones mostradas en la Figura 1, se establece la relación de las áreas como:

$$A_{FA} \leq A_{FCA} \leq A_{FCP} \tag{7.2}$$

donde A_{FA}, A_{FCA} y A_{FCP} son las áreas de los cercos Adaptado, convexo adaptado y convexo poligonal, respectivamente.

Cualquiera de esas tres configuraciones podrían ser consideradas como referencia para realizar el análisis de la menor área del sector ocupado. Pero nosotros consideramos que el cerco adaptado es la opción más real por ser de área menor; ya que el objetivo del trabajo de tesis es considerar una configuración adecuada de estados compactados, ocupando menor área.

Si comparamos la relación de mejor configuración de compactación entre dos situaciones, cualquiera de una serie de situaciones $S_1, S_2, ..., S_n$ generadas por las iteraciones de compactación, veremos que no interesa cual de las situaciones de áreas (A_{FA}, A_{FCA} o A_{FCP}) determina la mejor situación. Sean S_i y S_j dos situaciones diferentes, cuyas áreas, según la expresión (7.3), son, respectivamente:

$$A^i_{FA} \leq A^i_{FCA} \leq A^i_{FCP} \quad \text{y} \quad A^j_{FA} \leq A^j_{FCA} \leq A^j_{FCP} \tag{7.3}$$

Si la situación S_i es mejor que la S_j, indicada por sus respectivas áreas de FA, $A^i_{FA} \leq A^j_{FA}$, no implica que necesariamente FCA o FCP indiquen que S_i sea mejor que S_j, ni viceversa. Con esto queremos indicar que ninguna de las tres situaciones es óptima respecto a las otras. En esta decisión, uno de los criterio de elección de las situaciones dependerá de la complejidad de su implementación, tanto respecto al tiempo, como al uso de recursos de memoria que estos demanden.

7.2.1. Cerco convexo adaptado

Consideramos cerco convexo adaptado a una región convexa limitada por un contorno que se adapta al comportamiento de los segmentos de contornos de objetos ocupando la región. El contorno de la región esta compuesto por elementos tangenciales (segmentos de rectas y puntos) definidos por una recta imaginaria que rota tangencialmente, siguiendo un sentido (horario o anti-horario), en torno de los objetos en situación compactada. De esta forma, el contorno del cerco tendrá segmentos de recta en partes de convexidad o objetos vecinos (recta tangente común a dos objetos), y puntos en las curvas de objetos, tal como muestra la Figura 7.2(a).

La definición geométrica del cerco convexo adaptado envolviendo objetos curvos irregulares aún es motivo de investigación, ya que no se conoce algún método en la literatura de geometría computacional que lo haya estudiado.

7.2.2. Cerco convexo poligonal

Consideramos cerco convexo poligonal al polígono que contiene a los objetos en situación de compactada, como muestra la Figura 7.2(b). El polígono se define con una muestra de puntos relacionados con la definición de los contornos de los objetos, en particular por un subconjunto de puntos de control de los objetos definidos por curvas B-splines cúbicas.

Dados m puntos en el plano, definidos por la unión de todos los puntos de control de los n objetos en situación compactada, que definen un cerco poligonal con h, $h \leq m$, puntos de control siguiendo una técnica eficiente de construcción del cerco convexo de m puntos en el plano.

La técnica para poder implementar este tipo de cerco puede ser Jarvis ya que demanda O(m log m) operaciones o la técnica de QuickHull ya que demanda O(m log m) operaciones [Preparata92][CarvFigue94].

El cerco definido por los puntos de control de los objetos definidos por curvas B-splines cúbicas también contiene a los objetos en situación compactada, justificado por el siguiente principio.

Principio: El cerco convexo de los puntos de control de objetos definidos por curvas B-splines cúbicas contiene a los objetos compactados.

Prueba: Un objeto O^i es definido por la unión de segmentos de curva B-splines cúbicas S^i_j generadas por la interpolación de cuatro puntos de control adyacentes $A^i_j = C^i_{j-1}C^i_jC^i_{j+1}C^i_{j+2}$.

Por una propiedad conocida de curvas B-splines cúbicas [Roger85], el segmento de curva S^i_j está incluido por el cerco convexo de los puntos de control A^i_j. Como el contorno de O^i está definido por n segmentos, $\{S^i_j\}_{j=1,..,n}$, entonces la unión de los n cercos convexos respectivos definen dos polígonos concéntricos de $\frac{n}{2}$ vértices cada uno, el interior P^i_{int} y exterior P^i_{ext}. El contorno de O^i estará limitado por los polígonos P^i_{int} y P^i_{ext}. Siendo P^i_{ext} que contiene a O^i. Para m objetos, $\{O^i_j\}_{i=1,..,m}$, sus respectivos polígonos, $\{P^i_{ext}\}_{i=1,..,m}$, están contenidos en el cerco convexo de los vértices de $\{P^i_{ext}\}_{i=1,..,m}$; por tanto, los m objetos están contenidos por tal cerco convexo.

Axioma: Un conjunto de polígonos está contenido por el cerco convexo que con-

forman sus vértices.

7.2.3. Cerco Adaptado

Consideramos cerco adaptado al contorno cerrado definido por la unión de segmentos de contornos externos de los objetos externos de una situación de objetos compactados. Los segmentos de contorno de un objeto se unen al segmento del contorno cuando existe una proximidad de sus extremos. La Figura 7.2(c) muestra un ejemplo de esta categoría de cerco. La frase "adaptada"se debe a que el cerco es adaptado a segmentos de objetos externos de la situación compactada, siendo el cerco adaptado a la situación.

Una forma de determinar un cerco adaptado aproximado es utilizando el método de partición por árbol cuaternario (quatree) [Foley85], el cual divide cada cuadrante mientras exista la intersección de los contornos con los cuadrantes.

La intersección de los cuadrantes con los contornos de los objetos será verificada por la intersección del cuadrando, considerado como una caja rectangular isotética, con sub-árbol de cajas orientadas asociados a los objetos. Así, no se verificará todos los objetos, sino solamente el árbol de objetos que inicialmente interceptaba con el nodo superior del árbol. Un cuadrante, o nodo del quatree, será: a) dividido en cuatro cuadrantes iguales; b) será considerada como hoja válida; o c) será considerada una hoja inválida. Un cuadrante será dividido en cuatro cuadrantes iguales si existe un segmento considerable de contorno en su interior, caso contrario puede considerarse hoja inválida si no existe intersección con el contenido del interior del objeto. Será considerada hoja válida si la mayor parte del cuadrante es interior de un objeto.

7.3. Algoritmos de Cerco Convexo Bidimensional

El cálculo del cerco convexo bidimensional de un conjunto de puntos ha sido sin duda uno de los problemas a la vez más básicos y más ampliamente estudiados a lo largo de la historia de la Geometría Computacional. Además, los campos en los que ha resultado un cálculo útil sorprenden por lo aparentemente poco relacionados con las matemáticas en general y la geometría en particular. Tal es el caso del reconocimiento

de patrones, procesamiento de imágenes, etc.

En la literatura podemos encontrar diversos algoritmos que calculan el cerco convexo de un conjunto de puntos en el plano; sin duda es uno de los planteamientos de la Geometría Computacional que más soluciones ha conseguido. Desde incluso antes del nacimiento oficial de esta disciplina se conoce el trabajo de Graham en el año 1972. Se han propuesto estrategias siguiendo los paradigmas más empleados como el método incremental y el divide y vencerás. Todos estos algoritmos construyen el cerco convexo en tiempo $O(n \log n)$.

A continuación presentamos algunos de estos algoritmos que constituyen el conjunto clásico de algoritmos para la resolución del cerco convexo bidimensional, al que habría que añadir otras técnicas posteriores.

7.3.1. Algoritmos Incrementales

Algunos de los algoritmos que presentan complejidad $O(n \log n)$ son increntales, tal así presentaremos a continuacion algunos de estos algoritmos.

Algoritmo de Graham

El algoritmo de Graham (1960), se basa en la idea de que es posible determinar, de modo eficiente, los puntos extremos [FC91] de un conjunto de puntos en el plano formando un poligono estrellado el cual sería la primera aproximación del cerco convexo, para conseguir esto, se debe de realizar previamente una ordenación de los puntos alrededor de un punto interior a su cerco convexo el cual no ofrecería mayor problema. El poligono estrellado para un conjunto de puntos se muestra en la Figura 7.3 (a).

El metodo de Graham funciona de la siguiente manera: Al inicio se debe calcular la envolvente convexa superior ordenando de izquierda a derecha, es decir en orden creciente de abscisas. Así se obtiene una poligonal. Luego cada vértice de esa poligonal que quede por debajo del segmento que une el vértice anterior y el vértice posterior a él, se elimina uniendo éstos para formar una poligonal con un vértice menos. Cuando esta operación no pueda realizarse más habremos obtenido la envolvente convexa superior. La envolvente inferior se calcula de forma análoga, eliminando esta vez los

puntos que estén por encima del segmento que une el anterior y el posterior. De esta manera conseguida la poligonal obtenemos el cerco convexo deseado como podemos observar en la Figura 7.3 (b).

El orden de ejecución de este algoritmo viene dado por la necesidad de ordenar los puntos angularmente. Cualquier ordenación necesita al menos un tiempo $O(n \log n)$. A este tiempo se le debe añadir el del proceso ideado por Graham. Para averiguar este tiempo basta con percatarse que en cada paso un punto es tenido en cuenta o es descartado. En el peor de los casos, cuando la envolvente convexa es un triángulo, necesitamos del orden de $2n - 3$ operaciones.

Todo lo anterior nos indica que Graham es un algoritmo que trabaja en tiempo óptimo $O(n \log n)$

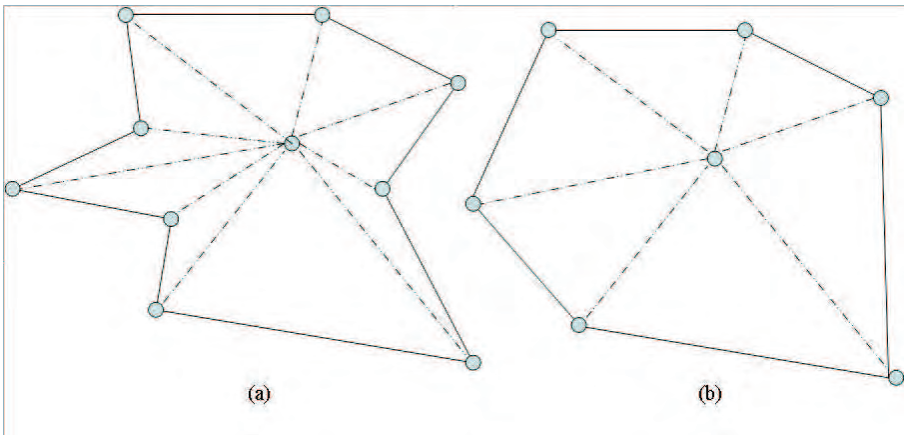


Figura 7.3: Funcionamiento Algoritmo de Graham

Algoritmo de Jarvis

El método de Jarvis (1973) es tambien un algoritmo incremental, el cual presenta una etapa de pre-procesamiento, su complejidad depende del número de puntos de entrada. La idea de este algoritmo consiste en imaginar una recta que se desplaza desde menos infinito hacia arriba hasta tocar un punto del conjunto de puntos. Luego, dicha recta va rodeando la nube de puntos en sentido positivo, girando en cada vértice hasta tocar el siguiente, de modo que al final la envuelve por completo.

La figura 7.4 muestra como se va realizando el desplazamiento de la recta indicada anteriormente para conseguir el cerco convexo.

Como indicamos anteriormente, es fácil de intuir que este proceso tardará en ejecutarse un tiempo proporcional al número de puntos en la envolvente y al tamaño de la nube. Cada barrido angular necesita un tiempo de proceso del orden de $O(n)$ pasos. Si la envolvente final posee k puntos, el tiempo final de ejecución es del orden de $O(nk)$. En el mejor de los casos el comportamiento puede ser lineal, por ejemplo si la envolvente es un triángulo. El peor de los casos, sin embargo, se da cuando todos los puntos están en la envolvente convexa, convirtiéndose en un método cuadrático.

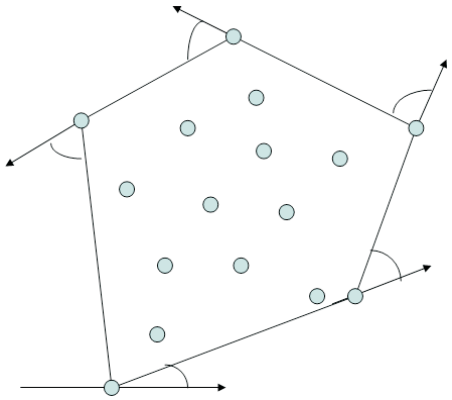


Figura 7.4: Algoritmo de Jarvis.

Algoritmo de Andrew

El método de Andrew es una variación del algoritmo de Graham donde la forma de ordenación de los puntos se determina ubicando los puntos de mayor y menos abcisa en la nube de puntos para luego trazar una recta que pase por ellos. Este algoritmo presenta complejidad $O(n \log n)$.

7.3.2. Algoritmos Dividir para Conquistar

Ahora examinaremos algoritmos que usan, de forma explícita, el paradigma de dividir para conquistar. Se busca dividir un problema en subproblemas del mismo tipo y aproximadamente todos ellos del mismo tamaño, resolver los subproblemas recursivamente y, finalmente, combinar la solución de los subproblemas para dar una solución al problema original. La recursión finaliza cuando el problema es pequeño y la solución es fácil de construir directamente [FC91].

La idea básica para calcular el cerco convexo bidimensional de los algoritmos del tipo dividir para conquistar es dividir un conjunto de puntos C en conjuntos menores $C1$ y $C2$, obteniendo los cercos convexos $conv(C1)$ y $conv(C2)$, para luego combinarlos de modo de obtener $conv(C)$. El paso mas importante aquí es el de combinación, donde, dados los polígonos convexos $P1$ y $P2$, deseamos obtener $conv(P1 \approx P2)$ [FC91]. A continuación examinaremos dos algoritmos que implementan estas ideas. Estos algoritmos son QuickHull y MergeSort.

Algoritmo QuickHull

Este algoritmo fue propuesto por Preparata y Shamos en 1985 [PS85]; fue denominado Quickhull por su similitud con el método de ordenación Quicksort. La idea básica del algoritmo es que para la mayoría de las configuraciones de puntos que se pueden presentar, se puede en cada paso descartar muchos puntos a la hora de construir la envolvente convexa ya que usa recursividad lo cual disminuye el tiempo computacional [RGV01].

El primer paso del algoritmo de quickhull es realizar la etapa de preprocesamiento, en la cual se requiere encontrar cuatro puntos extremos en las direcciones norte, sur, este y oeste y formar un cuadrilátero que estos puntos definen (este cuadrilátero puede ser degenerado); ahora todos los puntos en el interior de dicho cuadrilátero no son extremos, con lo cual pueden ser descartados. Así quedan puntos repartidos en cuatro regiones no conectadas entre si, se trata luego cada una de estas regiones independientemente donde se debe de encontrar el cerco convexo contenido en las cuatro regiones triangulares exteriores al cuadrilátero.

En cada una de estas regiones se buscara el punto más alejado a la recta que define dicha región, obteniendo así cuatro nuevos puntos y un polígono de a lo más ocho lados que divididirá a los puntos que no hemos eliminado en ocho regiones que tratamos individualmente siguiendo la misma regla anterior. Sucesivamente se van dividiendo las regiones resolviendo el problema recursivamente, obteniendo al final el cerco convexo bidimensional buscado. La Figura 7.5 mostrada a continuación muestra la búsqueda del punto mas alejado c, consiguendose formar un triangulo unido por los puntos a,b y c.

Este algoritmo presenta complejidad $O(nlogn)$ en el mejor de los casos cuando los

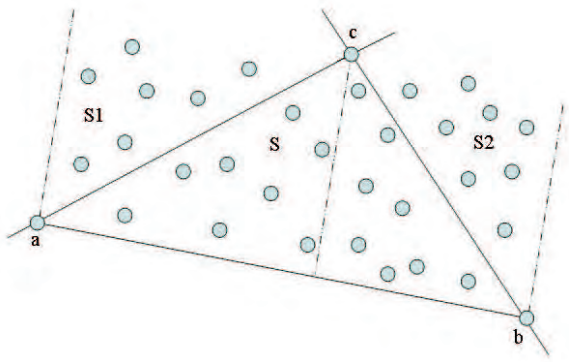


Figura 7.5: Tratamiento de Algoritmo de QuickHull.

puntos se encuentran distribuidos uniformemente en el plano y complejidad $O(logn^2)$ en el peor de los casos.

Algoritmo MergeHull

El algoritmo MergeHull es tambien del tipo dividir para conquistar, al igual que el quickhull debe su nombre a la semejanza con el algoritmo de mergesort; este algoritmo presenta complejidad $O(n \log n)$. La etapa mas crítica de este algoritmo es la de combinación donde los cercos convexos obtenidos por los subconjuntos $S1$ y $S2$ son combinados para producir el cerco convexo de S.

Este algoritmo divide el problema en diversos subproblemas similares al problema original pero de tamaño menor. El algoritmo procede a resolver éstos subproblemas, luego realiza la combinación de sus soluciones, obteniendo una solución del problema para su instancia original.

En su primera fase el algoritmo divide un conjunto de puntos S en subconjuntos $S1$ y $S2$, esta división es realizada en relación a la mediana que forman las coordenadas en el eje x y puede ser vista como una separación en realación a una recta vertical. El conjunto de puntos $S1$ será formado por los puntos que tienen coordenada en x menor o igual a la mediana de los puntos en x de los puntos de S. El conjunto $S2$ será formado por los demás puntos de S. Si existe alguna repetición de coordenada de x de algun punto, entonces se considera una ordenación de esos puntos por los valores de las coordenadas en y.

En la siguiente fase el algoritmo procede a resolver individualmente cada subproblema mediante recursión, hallando el cerco convexo para cada subconjunto de puntos. En su fase final se trata de encontrar dos rectas tangentes a $conv(S1)$ y $conv(S2)$, a partir de estas resultara fácil construir $conv(S1 \bigcup S2)$ en tiempo lineal.

En la Figura 7.6(a) tenemos dos polígonos los cuales tienen sus vértices ordenados en forma polar en torno de un punto interior común $p0$. En la figura 7.6(b). El polígono $S2$ está contenido en un ángulo de vértice $p0$ y definido por las semi-rectas $p0a$ y $p0b$, donde a y b son vértices de $S2$. Los vértices a y b separan la lista de vértices de $S2$ en dos sublistas: una ordenada siguiendo ángulos polares (en torno de $p0$) crecientes y otra ordenada siguiendo ángulos polares decrecientes. La segunda lista correspondiente a los vértices de $S2$ que son interiores al triángulo $p0ab$ y puede, por tanto, ser descartado. La primera lista puede ser combinada como la lista de los vértices de $S1$ en tiempo $O(n)$, donde n es el número total de vértices de $S1$ y $S2$.

Asímismo tambien podemos realizar la combinación de los dos polígonos convexos mediante la técnica de tangentes. Según Rocha et al [RGV01] las tangentes son formadas por los segmentos entre los vertices a y b (puntos que forman el cerco convexo de $S1$ y $S2$ respectivamente) y los puntos más a la derecha de $conv(S1)$ y más a la izquierda de $conv(S2)$. Una vez halladas las tangentes se procede a recorrer los cercos convexos en sentidos opuestos verificándose si se puede realizar un intercambio entre los vértices. La búsqueda terminará cuando ninguno de los puntos tiene que ser intercambiado por otro.

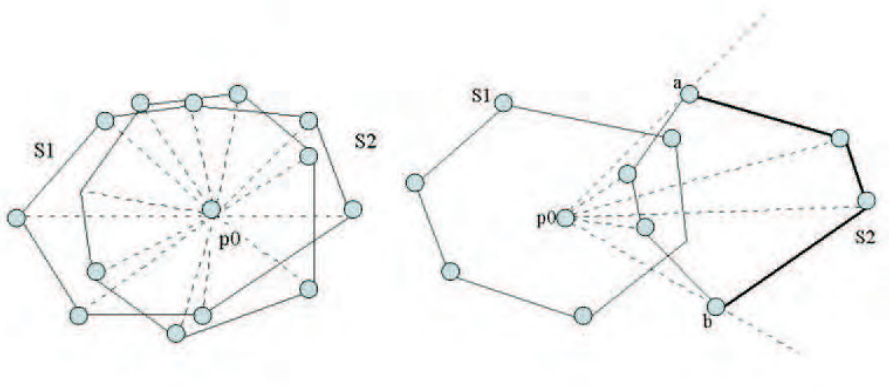


Figura 7.6: Algoritmo de MergeHull

Analisando el funcionamento de los algoritmos descritos anteriormente, es posible notar una característica común a ambos la cual es la presencia de una etapa de

ordenación como pre-procesamiento es así que el mayor esfuerzo, en términos de timpo computacional, ocurre al momento de realizar la ordenación de puntos.

7.4. Pseudocódigo del Módulo analizador de áreas

A continuación se muestra el diseño del módulo analizador de areas, el cual implementa un algoritmo Quickhull para determinar el cerco convexo bidimensional que conforman los objetos en estado de compactación dentro de una superficie contenedora. Se desarrollará el algoritmo principal del módulo analizador de areas así como cada uno de los submódulos que lo conforman, mostrando su funcionamiento y pseudocódigo en cada parte.

En el Cuadro 7.1 se muestra una descripción de las variables a utilizar.

Variables a usar en el modulo Analizador de Areas	
Variable	**Descripción**
sysobjs	Estructura que contiene las caracateristicas de los objetos
convexHull	Estructura que contiene los vértices que forman el cerco convexo
listaPtos	Conjunto de todos los puntos de control determinados por cada objetos
areaCerco	Area conformada por el cerco convexo

Cuadro 7.1: Variables del módulo analizador de áreas.

7.4.1. Módulo Principal

Funcionamiento

Se obtiene la lista de puntos haciendo uso de la función $CargarPtosFecho$, la cual recibe como parámetro una estructura $sysobjs$ que contiene las caracteristicas de los objetos, para luego realizar el calculo del cerco convexo haciendo uso de la funcion $QuickHull$ que recibe como parámetro la lista de puntos. A continuacion el modulo procede a calcular el área determinada por el cerco convexo haciendo uso de la funcion $CalculaAreaFecho$.

Pseudocódigo

1: $listaPtos \leftarrow CargaPtosFecho(sysobjs)$
2: $convexHull \leftarrow QuickHull(listaPtos)$
3: $areaFecho \leftarrow CalculaAreaFecho(convexHull)$

Comentario

En el punto 2 se realiza del calculo del cerco convexo que determina el conjunto de puntos que se esta pasando como parametro, se hace uso de un algoritmo de QuickHull para poder determinar el conjunto de vertices que conforman el cerco convexo que estamos buscando.

7.4.2. CargaPtosFecho

Funcionamiento

La función $CargaPtosFecho$ se encarga de obtener todos los puntos de coordenadas x, y, z que son determinados por los objetos en el ambiente de simulación. Recibe como parámetro una estructura $sysobjs$ la cual contendrá las caracteristicas de los objetos luego de la compactación. Mientras existan objetos dentro de la estructura $sysobjs$ va cogiendo las coordenadas de cada uno y las va pasando a la estructura $listaPtos$.

Pseudocódigo

1: $i \leftarrow 1$
2: **while** $i \leq sysobjs.nobjs$ **do**
3: $ob \leftarrow sysobjs.objs[i]$
4: **for** $j = 0$ to $ob.npc$ **do**
5: $p \leftarrow ob.pc[j]$
6: $Insertar(p, listaPtos)$
7: **end for**
8: $i \leftarrow 1$
9: **end while**

Comentario

En el punto 2 la variable $sysobjs.nobjs$ determina el número de objetos compactados que se están procesando así como en el punto 4 se procesara cada punto de control que determina el objeto $sysobjs.objs[i]$.

7.4.3. QuickHull($listaPtos$)

Esta función construye el cerco convexo a partir de un conjunto de puntos $listaPtos$.

Funcionamiento

Esta funcion procede a encontrar el punto (a) más a la izquierda del conjunto $listaPtos$, luego de manera similar procede a encontrar el punto (b) más a la derecha del conjunto $listaPtos$, luego se traza una recta y se divide el conjunto $listaPtos$ en dos subconjuntos; para los cuales se hallará su cerco convexo llamando usando una función recursiva $QHull$, una vez conseguidos los cercos convexos de los dos subconjuntos se procede a concatenarlos mediante la función $concatenarQHull$ con lo cual se obtiene el cerco convexo $convexHull$.

Pseudocódigo

1: Sea a el punto más a la izquierda de $listaPtos$
2: Sea b el punto más a la derecha de $listaPtos$
3: Sea $listaPtos_1$ el conjunto de puntos a la izquierda de la recta $a - b$
4: Sea $listaPtos_2$ el conjunto de puntos a la derecha de la recta $a - b$
5: $convexHull_1 \leftarrow QHull(listaPtos_1, a, b)$
6: $convexHull_2 \leftarrow QHull(listaPtos_2, b, a)$
7: $convexHull \leftarrow concatenarQHull(convexHull_1, convexHull_2)$

Comentario

Para el punto 3 y 4 el algoritmo procede a encontrar una recta $a - b$ la cual pueda separar en dos subconjuntos al conjunto de puntos $listaPtos$.

7.4.4. QHull($S, punto1, punto2$)

Funcionamiento

Esta funcion recursiva procura primeramente encontrar un punto c más distante de la recta $a - b$, luego calcula recursivamente el cerco convexo de los puntos a la izquierda de $a - c$, así como el cerco convexo de los puntos a la derecha de $c - b$. Finalmente se realiza una combinación de los dos cercos convexos encontrados. El caso base se da cuando no existe ningun punto a la izquierda de $a - c$, con lo cual el algoritmo retorna esta arista.

Pseudocódigo

1: **if** $S = \{a, b\}$ **then**
2: retornar el segmento orientado $a - b$
3: **else**
4: Sea c un punto de S tal que la distancia entre c y la recta $a - b$ sea máxima.
5: Sea S_1 el conjunto de puntos de S a la izquierda de $a - c$
6: Sea S_2 el conjunto de puntos de S a la izquierda de $c - b$
7: $convexHull_1 \leftarrow QHull(S_1, a, c)$
8: $convexHull_2 \leftarrow QHull(S_2, c, b)$
9: $convexHull \leftarrow concatenarQHull(convexHull_1, convexHull_2)$
10: **end if**

Comentario

En el punto 7 y 8 se va calculando recursivamente el cerco convexo de cada subconjunto de puntos. En 1, el conjunto de puntos S se va actualizando de tal manera que al final solo quedará el segmento orientado $a - b$.

7.4.5. CalculaAreaFecho(convexHull)

Funcionamiento

Esta función realiza el cálculo del area del cerco convexo a partir del conjunto de vertices $convexHull$ hallados por el algoritmo de QuickHull. Se van buscando aristas y hallando parcialmente el area de los triángulos que forman tales aristas con el centro de masa $centm$.

Pseudocódigo

1: $centm \leftarrow CalcularCentroMasaFecho(convexHull)$
2: $nodo \leftarrow ObtenerPunto(convexHull)$
3: $areafecho \leftarrow 0$
4: $verticef1[0] \leftarrow nodo.x - centm[0]$ {calculamos el primer vértice}
5: $verticef1[1] \leftarrow nodo.y - centm[1]$
6: $verticef1[2] \leftarrow nodo.z - centm[2]$
7: $nodo \leftarrow ObtenerPunto(convexHull)$ {se obtiene el próximo punto}
8: **while** $nodo \neq \emptyset$ **do**
9: $verticef2[0] \leftarrow nodo.x - centm[0]$ {se obtiene el siguiente vértice}
10: $verticef2[1] \leftarrow nodo.y - centm[1]$
11: $verticef2[2] \leftarrow nodo.z - centm[2]$
12: $area \leftarrow calculareareatriangulo(verticef1, verticef2)$
13: $areafecho = areafecho + area$
14: $nodo \leftarrow ObtenerPunto(convexHull)$
15: $verticef1[0] \leftarrow verticef2[0]$ {Se hace una copia del vértice}
16: $verticef1[1] \leftarrow verticef2[0]$
17: $verticef1[2] \leftarrow verticef2[0]$
18: **end while**

Comentario

En el punto 1 se calcula el centro de masa del cerco convexo, en 2 se obtiene en la variable *nodo* las características de un punto de la estructura $convexHull$, en 4 se calcula el primer vertice del cerco, en 9 se obtiene otro vertice con lo cual en 12

se procede a calcular el área del triángulo formado por tales vértices para luego ir acumulando esta área en el punto 13 en la variable $areafecho$ la cual irá acumulando el área final del cerco convexo.

7.4.6. Ejemplos de Ejecucion del Algoritmo

A continuación presentaremos algunos ejemplos de la corrida del módulo analizador de áreas en la cual se mostrará el cerco convexo que envuelve a los objetos compactados.

Caso 1: Cerco convexo para cinco objetos compactados

Se trato cinco objetos en estado de compactación, para los cuales se procedió al cálculo de su envolvente convexa como muestra la Figura 7.7; los objetos presentaron las siguientes características:

Objeto	**Cantidad**	**Area**
Objeto1	1	249.77
Objeto2	2	231.73
Objeto3	2	197.35
Total Objetos		**1107.95**
Area Usada		**1529.87**

Cuadro 7.2: Valores de objetos compactados para el caso 1.

Como podemos observar en la tabla 7.2 el area total de los objetos es A_o =1107.95. El cálculo de la envolvente convexa dio como resultado un área de A_s =1529.87, con lo cual podemos calcular el valor del area de desperdicio A_d = 421,92 , satisfaciendo la Ecuacion 7.3.

Caso 2: Cerco convexo para siete objetos compactados

Se trato siete objetos en estado de compactación, para los cuales se procedió al cálculo de su envolvente convexa como muestra la Figura 7.8; los objetos presentaron las siguientes características, mostrados en el cuadro 7.3

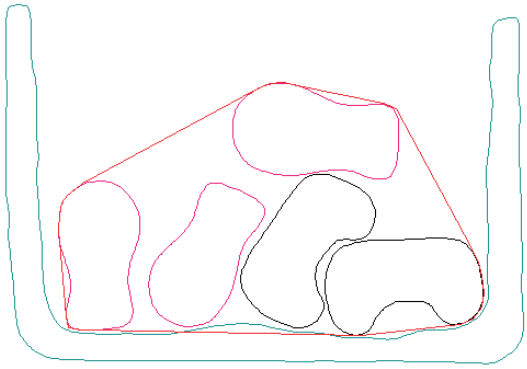


Figura 7.7: Area del sector ocupado para caso ejemplo 1.

Objeto	**Cantidad**	**Area**
Objeto1	1	249.77
Objeto2	1	231.73
Objeto3	5	197.35
Total Objetos		**1107.95**
Area Ocupada		**2160.76**

Cuadro 7.3: Valores de objetos compactados para el caso 2.

Como podemos observar en el Cuadro 7.3 el área total de los objetos es A_o =1468.27. El cálculo de la envolvente convexa dio como resultado un área de A_s=2160.76, con lo cual podemos calcular el valor del area de desperdicio A_d = 692,49, satisfaciendo la Ecuación 7.3 anteriormente citada.

Caso 3: Cerco convexo para cinco objetos compactados

Se trato cinco objetos en estado de compactación, para los cuales se procedió al cálculo de su envolvente convexa como muestra la Figura 7.9; los objetos presentaron las siguientes características:

Como podemos observar en la tabla 7.4 el area total de los objetos es A_o =1107.95. El cálculo de la envolvente convexa dio como resultado un área de A_s =1529.87, con lo cual podemos calcular el valor del area de desperdicio A_d = 421,92 , satisfaciendo la Ecuación 7.3 anteriormente citada.

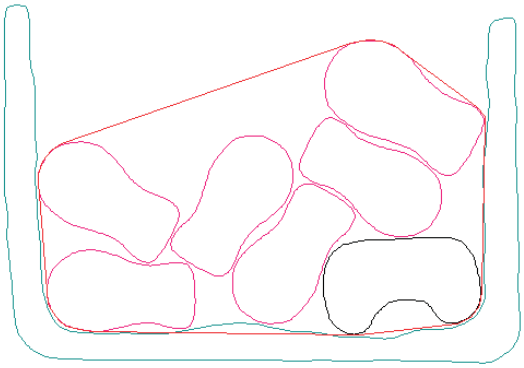


Figura 7.8: Area del sector ocupado para caso ejemplo 2.

Objeto	**Cantidad**	**Area**
Objeto1	1	249.77
Objeto2	2	231.73
Objeto3	2	197.35
Total Objetos		**1107.95**
Area Usada		**1529.87**

Cuadro 7.4: Valores de objetos compactados para el caso 3.

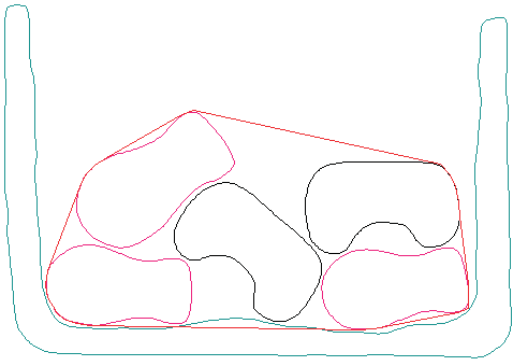


Figura 7.9: Area del sector ocupado para caso ejemplo 3.

Capítulo 8

Un Sistema SimulaCort para resolver el problema de cortes irregulares

En este capítulo se describirá el sistema que implementa el algoritmo grasp con simulación dinámica para resolver el problema de cortes irregulares propuesto en el Capítulo 4. En la Seccion 1, se describirá el sistema, en la Seccion 2 se revisará la implementación del sistema, mostrándose algunos extractos del código fuente. Finalmente, en la Seccion 3 revisaremos la configuración del hardware y software mínimo necesario para que el sistema funcione correctamente.

8.1. Descripción del Sistema

El sistema desarrollado implementa una técnica de algoritmo grasp con simulación dinámica, con lo cual se da solución al problema de cortes en una dimensión. Se encuentra básicamente constituído por la implantación del algoritmo grasp desarrollado en el Capítulo 4. Presenta módulos diferenciados tanto para la adquisicion de datos, procesamiento, visualización y salida de datos los cuales han sido descritos en el Capítulo 3 de arquitectura del sistema.

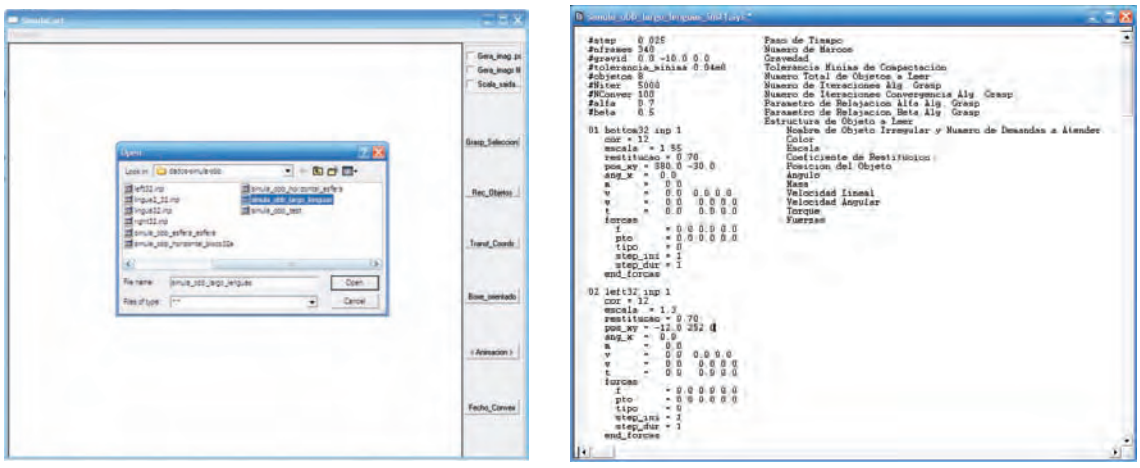
8.2. Módulos del Sistema

Se presentará cada módulo que se describio en el Capítulo 3 de arquitectura del sistema, describiendo su funcionalidad y su pantalla asociada.

8.2.1. Módulo de Lectura de Datos

Este módulo carga los datos en las estructuras correspondientes para luego ser manejadas por el sistema.

Aquí podemos encontrar la lectura de todos los parametros necesarios para el funcionamiento del sistema; se leen de un archivo de texto los objetos a cortar con sus respectivas demandas de corte, la superficie contenedora, las características propias de cada objeto, así como sus caracteristicas físicas, los parámetros para la ejecución de la simulación como son el número de frames, el paso de tiempo, etc. La Figura 8.1(a) muestra la carga de objetos desde un archivo plano. La estructura del archivo de datos de entrada presenta dos partes; una parte general referida a los parámetros de la simulación dinámica y del algoritmo Grasp y otra parte donde estan las caracteristicas de los objetos. En la Figura 8.1(b) mostramos dicha estructura indicando el significado de cada parámetro así como la estructura de cada objeto a ser leido.



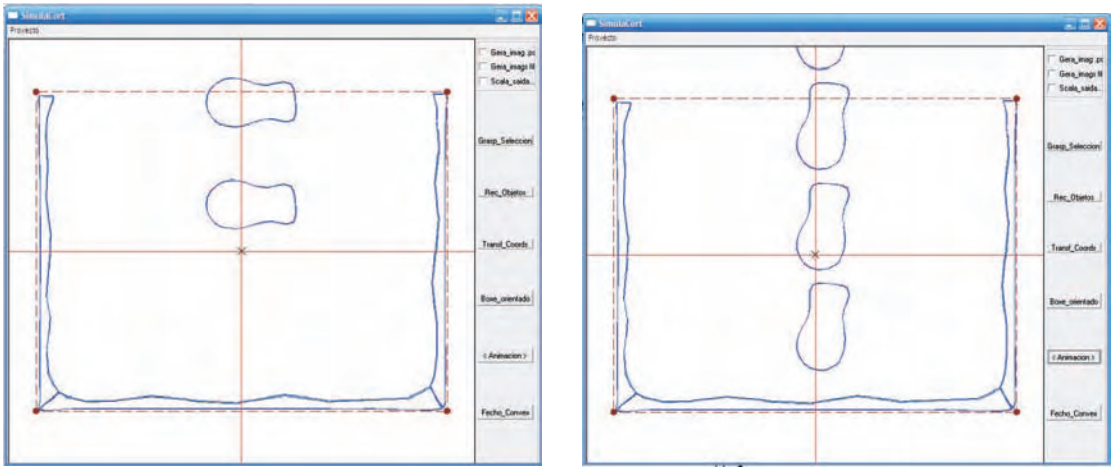
(a) (b)

Figura 8.1: Entrada de datos: (a) pantalla de carga de objetos; (b) estructura del archivo de datos de entrada.

8.2.2. Módulo de Procesamiento

Este módulo se encarga del procesamiento de las funcionalidades del sistema, las cuales van desde la metaheurística de seleccion, distribución de objetos, compactación y análisis de áreas.

La Figura 8.2(a) muestra la distribucion de los objetos dentro de la superficie siguiendo la técnica vertical. La Figura 8.2(b) muestra un instante en el cual se está realizando la simulación dinámica.



(a) (b)

Figura 8.2: Distribución de los objetos en el contenedor: (a) pantalla de distribución de objetos en la superficie; (b) pantalla de simulacion dinámica.

La Figura 8.3 muestra el cerco convexo resultante luego de que los objetos estan compactados.

8.2.3. Módulo de salida

Este módulo se encarga de realizar la presentación de los resultados, en el cual se muestran las superficies contenedoras finales a utilizar, cada una con su respectiva lista de objetos compactados.

La Figura 8.4 muestra la configuracion final de los objetos en su superficie contenedora.

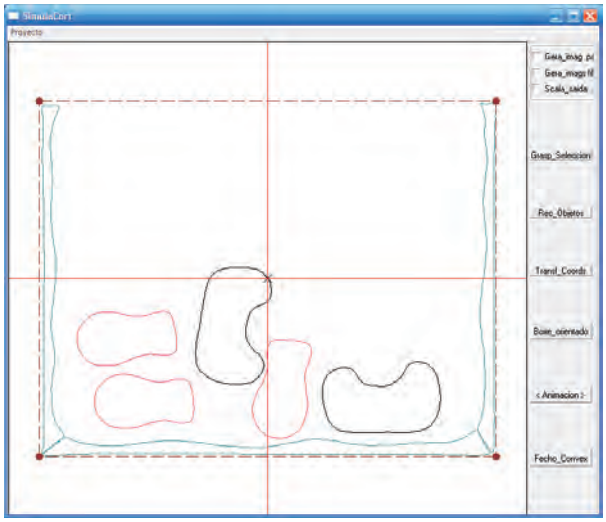


Figura 8.3: Pantalla de Análisis de Areas Finales

8.3. Implementación

El desarrollo del Sistema SimulaCort ha sido realizado en lenguaje C con Microsoft Visual Studio v.6; las librerias gráficas usadas son IUP/LED, CD las cuales han sido desarrolladas por el grupo de tecnologías gráficas (tecgraf) de la puc-rio, Brasil (www.tecgraf-rio.br) ya que este Ide está basado en metodología orientada a objetos y presenta características de división del programa en módulos; lo cual ayuda a un completo ordenamiento del programa. Se explicarán características referentes a la modularidad y la programación orientada a objetos, así como se mostrarán extractos del código fuente de las principales funcionalidades implementadas del sistema.

El sistema GenCort presenta varios módulos de programación, entre los mas importantes tenemos:

- bsplines - calculos - contato - dinamica - dspwindow - dwspline - graspseleccion - fechoconvexo - fileoper - impulso - inercia - mainspli - memorias - numerica - obb3tree - overlappairs - distribucionobjs

8.3.1. Implementación del Modulo fileoper

Este módulo de programación implementa el código fuente que permite la lectura de los parámetros para el funcionamiento del sistema. La función principal de

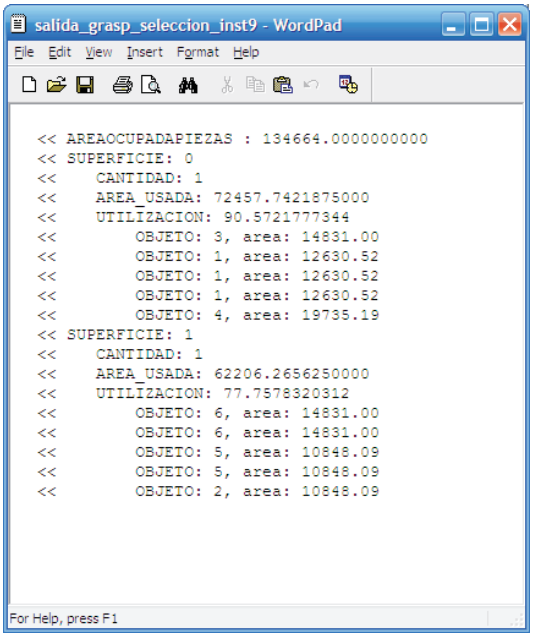


Figura 8.4: Pantalla de presentacion de resultados.

este módulo es *filcargadados*, es aquí donde se cargan los datos en las respectivas estructuras del sistema.

En el Apéndice A podemos observar un extracto del código fuente de la función *filcargadados*.

8.3.2. Implementación del Modulo graspseleccion

Este módulo de programación implementa el código fuente que permite procesar el algoritmo grasp seleccion y mejoria. La funcion principal de este modulo es *graspSeleccion*, es aquí donde se procesa el algoritmo grasp propiamente dicho y se obtienen las superficies con la lista de piezas a ser compactadas.

En el Apéndice A podemos observar un extracto del código fuente de la función *graspSeleccion*.

8.3.3. Implementación del Módulo distribucionobjs

Este módulo de programación implementa el código fuente que permite procesar la distribución inicial de los objetos en la superficie contenedora antes de aplicar

simulación dinámica. La funcion principal de este módulo es *distribuyeobjs*.

En el Apéndice A podemos observar un extracto del código fuente de la función *distribuyeobjs*.

8.3.4. Implementación del módulo dinámica

Este módulo de programación implementa el código fuente que permite procesar la simulación dinámica para compactar los objetos en sus respectivas superficies. La función principal de este módulo es *dinloopmovimiento*, es aquí donde se procesan iteración por iteración las escenas de movimiento de los cuerpos, detectándose posibles interferencias para aplicar impulsos.

En el Apéndice A podemos observar un extracto del código fuente de la función *dinloopmovimiento*.

8.3.5. Implementación del Módulo fechoconvexo

Este módulo de programación implementa el código fuente que permite procesar el análisis de las áreas finales de los objetos luego de la compactación con respecto a la superficie que los contiene. La función principal de este módulo es *quickHull*; es aquí donde se implementa el algoritmo de quickhull para poder hallar la menor envoltura que inscriba a los objetos compactados.

En el Apéndice A podemos observar un extracto del código fuente de la función *quickHull*.

8.4. Requerimientos de hardware y software

El equipo necesario para el funcionamiento adecuado del sistema diseñado es el siguiente:

Configuración de Hardware Mínimo: se necesita un equipo con las características que se mostrarán a continuación para el funcionamiento del sistema Simu-

laCort: Procesador Intel o Compatible; Velocidad mayor a 450 Mhz; Espacio en Disco: 12 Mb; Tarjeta de Video de 8Mb; Mouse; Teclado.

Configuración de Software Mínimo: Sistema Operativo Windows 9.X,NT,2000,XP, 2003; Editor de Textos.

Capítulo 9

Experimentos Numéricos

En este capítulo se mostrará el análisis del desempeño del sistema, presentándose las instancias de prueba. En la Sección 1, se indicará la configuración del equipo usado en las pruebas. En la Sección 2, se mostrará el set de problemas que se usarán para las pruebas. En la Sección 3, se mostrará en tablas resumen la ejecución de algunas instancias de prueba y sus resultados, para luego realizar en la Sección 4, un análisis de la eficiencia del algoritmo a partir de los datos obtenidos.

9.1. Hardware y Software empleado

Para los experimentos numéricos así como para el desarrollo del sistema se utilizó:

Hardware empleado: Máquina con procesador Intel Familia 6 Modelo 8, velocidad: 704 Mhz, memoria de 320MB. Se realizaron además pruebas del sistema en un computador Pentium II de 430 Mhz de Velocidad, con lo cual se verificó que el tiempo de procesamiento varía (aumento), al igual que el tamaño de la memoria y de la pila, por lo cual se recomienda como hardware óptimo a usar: Máquina con procesador Pentium III, 600 MHz de velocidad, memoria de 320Mb.

Software empleado: Visual Studio C++ 6.0, Windows XP Professional Edition, bibliotecas gráficas IUP/LED y CD de tecnologias gráficas (Tecgraf) PUC-Rio, Brasil.

Modo de video: Monitor 15", resolución de la pantalla de 800 x 600 píxeles, resolución del color de 32 bits.

9.2. Instancias de prueba

Las instancias de prueba, están conformadas por las demandas de corte de piezas irregulares (cantidad y tamaño de cada pieza de material) y la superficie contenedora que se tomará para alojar a las piezas. Se usarán 10 instancias de prueba diferentes. El número de piezas en demanda varía en cada caso según la instancia. Su especificacion se encuentra en el Apéndice B

9.2.1. Lista de Objetos a utilizar

El Cuadro 9.1 muestra los objetos que serán usados en nuestro set de pruebas, donde (a) es un objeto que representa la superficie contenedora compuesta por tres objetos (dos verticales y una base); los objetos que representan las piezas de corte son (b), (c) y (d), estos estan definidos por 32 puntos de control. Esas piezas pueden ser usadas o replicadas tantas veces se requiera, así como orientarlos y escalarlos a la forma deseada.

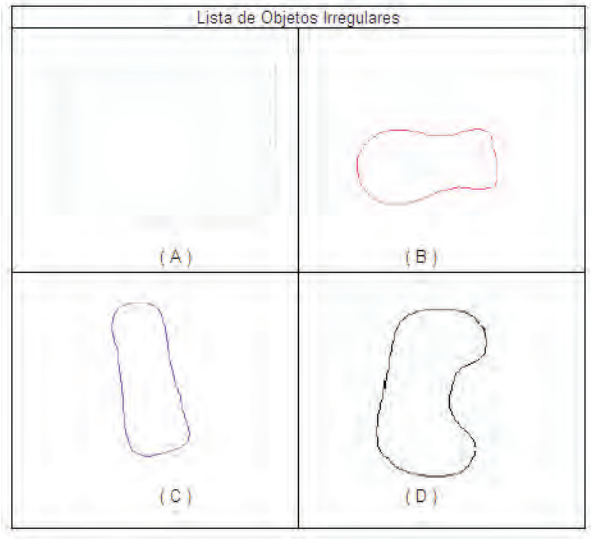


Figura 9.1: Objetos usados en la simulación. (a)Contenedor; (b)Objeto Irregular IDlingua32; (c)Objeto Irregular IDlet1; (d)Objeto Irregular IDlingua23

9.3. Experimentos numéricos

Aqui se presenta la ejecución del algoritmo sobre las 10 instancias de prueba descritas en la seccion anterior.

Se utilizo una superficie contenedora patrón la cual se le fue variando el tamaño a ocupar; este tamaño de ocupacion varia entre 40 % y 60 % como vamos a observar al momento que presentemos el resultado de las instancias de prueba.

9.3.1. Ejecución de las instancias

Para poder llevar a cabo la ejecución del algoritmo se necesitaron parametros grasp y de simulación dinámica como se muestran en las tablas a seguir presentadas.

Parámetros Algoritmo GRASP

Los parametros que se muestran a continuación son resultado de una previa calibración. Para los parametros de relajación GRASP ALFA y BETA se ejecuto el algoritmo tomando valores desde 0 a 1, como resultado se obtuvieron $ALFA = 0{,}7$ y $BETA = 0{,}5$

PARAMETRO	VALOR
Numero de Iteraciones	500
Convergencia	100
Parametro Relajacion ALFA	0.7
Parametro Relajacion BETA	0.5

Los parámetros de la simulación dinámica son mostradas en la tabla a seguir.

PARAMETRO	VALOR
Paso de Tiempo	0.025
Numero de Frames	340
Gravedad	0.0 -10.0 0.0
Tolerancia Minima	$0{,}04e^0$

Con esos parametros, se ejecutaron las simulaciones obteniendose los resultados. Aqui mostramos el resultado de la primera instancia de prueba; el resultado de las demas instancias de prueba pueden ser vistas en el Apéndice B.

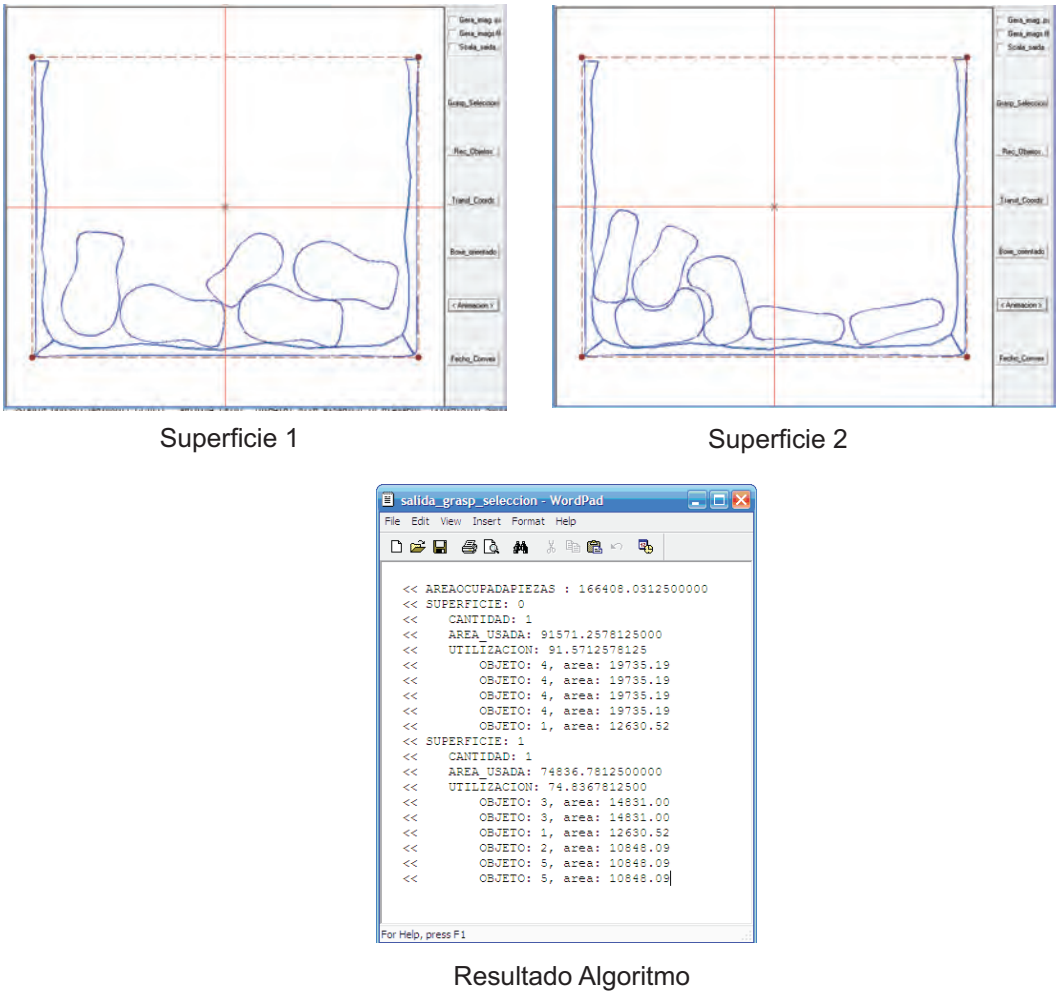


Figura 9.2: Figura Resultado Ejecución Instancia 1

Seguidamente, mostramos mediante una tabla un resumen del resultado de la ejecucion del algoritmo, presentando el número de superficies contenedoras utilizadas, la lista de objetos a posicionar, el porcentaje de error generado, el área utilizada y la solución óptima; así como la configuración final de cada superficie luego de la compactación.

El Cuadro 9.1 muestra la descripción de las columnas de las tablas.

Columna	Descripcion
Nombre	Nombre de la Instancia de Prueba
AreaSup	Area de la superficie contenedora
M	Objetos de Tamaño diferente
N	Cantidad total de piezas en demanda
NGEN	Número de Generaciones Algoritmo GRASP
SolGen	Cálculo del Número de Superficies Generadas por el algoritmo.
% Error	Porcentaje de Error encontrado respecto a la solución óptima.
AreaUti	Area total utilizada.
SolOpt	Número de Superficies utilizadas.

Cuadro 9.1: Tabla de descripción de las columnas de tablas usadas en la experimentos numéricos.

Resultado de las Pruebas								
Nombre	**AreaSup**	**M**	**N**	**NGEN**	**SolGen**	**Error**	**AreaUti**	**SolOpt**
instancia1	100 000	5	11	500	2	0 %	83.2 %	2
instancia2	90 000	5	11	500	2	0 %	92.44 %	2
instancia3	105 000	5	14	500	2	0 %	89.01 %	2
instancia4	105 000	5	14	500	2	0 %	95.78 %	2
instancia5	120 000	5	14	500	2	0 %	82.89 %	2
instancia6	120 000	6	16	500	2	0 %	90.24 %	2
instancia7	100 000	6	14	500	2	0 %	94.35 %	2
instancia8	90 000	6	11	500	2	0 %	83.56 %	2
instancia9	80 000	6	10	500	2	0 %	84.16 %	2
instancia10	120 000	6	16	500	2	0 %	91.16 %	2

Cuadro 9.2: Tabla de resultados de las pruebas

9.4. Análisis de la Eficiencia

Luego de realizar la ejecución de todas las instancias de prueba mostradas en la seccion anterior, se procedió a analizar la confiabilidad del sistema mediante el error promedio obtenido el cual es 0 %. Las instancias de prueba con las que se probo el algoritmo son pequeñas, con lo cual si aumentamos las demandas de corte, el algoritmo entregara soluciones de calidad.

Capítulo 10

Conclusiones y trabajos futuros

En este trabajo de tesis se ha propuesto un Algoritmo GRASP con Simulación Dinámica para resolver el Problema de Cortes irregulares, el cual tiene diversas aplicaciones en la industria. Se ha comprobado empíricamente que aplicando la metaheurística GRASP, se obtienen mejores resultados que los métodos tradicionales de la programación lineal entera, con respecto a conseguir la cantidad minima de superficies que alberguen una determinada cantidad de piezas irregulares en demanda. Para ello se ha desarrollado un sistema "SimulaCort"que implementa el Algoritmo GRASP con Simulacion Dinamica lo que constituye el principal aporte de este proyecto.

Por medio de la utilización de un Algoritmo GRASP hemos hallado soluciones de calidad a distintas instancias del problema planteado. Con el objetivo de justificar este buen comportamiento hemos analizado los resultados obtenidos, encontrando en nuestro algoritmo un bajo grado de error con respecto al area ocupada de las piezas en su respectiva superficie contenedora, pero ningun error en la determinación del minimo número de superficies.

Para comprobar la calidad de las soluciones proporcionadas por el algoritmo desarrollado, se ha probado una serie de instancias de prueba, comparando la calidad de las soluciones que ofrece nuestro algoritmo con respecto a las soluciones óptimas; comprobándose que el algoritmo propuesto tiene un funcionamiento eficiente consiguiendo en todos los casos llegar a la solución óptima.

Existen muchos algoritmos y métodos con diferentes factores que son tomados en cuenta para resolver el problema de cortes irregulares; pero el más importante de ellos

es el de minimización de la pérdida de material. En este presente trabajo se ha logrado este objetivo con un bajo grado de error con respecto al porcentaje de utilización de la superficie contenedora, consiguiendo desarrollar un algoritmo GRASP con Simulación Dinámica sencillo, robusto y fácil de usar.

Frente a los métodos existentes planteados en la literatura, surgio la necesidad de construir un método el cual trate de dar solución al problema de cortes irregulares de manera eficiente. El método que hemos desarrollado implementa un algoritmo el cual es una alternativa promisoria que puede superar a los métodos existentes; este método esta constituido por técnicas de animaciones basadas en leyes físicas con análisis de contactos y colisiones [Ri01], [RCV02] para simular la compactación de los pedazos de cortes; la generacion de cada iteracion se realizo con un mecanismo inteligente combinada con la tecnica GRASP [Ma03].

Debido a que el tiempo de procesamiento es un factor de vital importancia, las instancias de prueba fueron escogidas de tal manera, que las cantidades de piezas en demanda no sean muy grandes; ya que el algoritmo GRASP con Simulacion Dinamica desarrollado, esta orientado a resolver de manera rápida y eficiente el problema de cortes irregulares considerando objetos en demanda de forma irregular. El algoritmo resuelve casos cuando las demandas son mayores, pero su tiempo de procesamiento es mayor, por lo cual si se aumenta en forma excesiva el número de demandas el tiempo de procesamiento crecería exponencialmente, debido al modulo de simulacion dinamica, el cual se muestra graficamente.

En el presente trabajo se logró configurar adecuadamente los parámetros del algoritmo Grasp y de Simulacion Dinamica, siendo de gran importancia que estos se encuentren debidamente calibrados para que se consigan soluciones cada vez mejores. Como resultado se obtuvo un equilibrio entre la eficacia y la eficiencia. Este equilibrio es configurable mediante los parámetros y operaciones usados en el algoritmo.

En el desarrollo del presente trabajo se muestra una aplicación directa del área de la Inteligencia Artificial en la industria. La metaheuristica GRASP con Simulacion Dinamica ha demostrado poder resolver problemas difíciles catalogados con la característica NP, por lo cual es necesario que los investigadores en esta rama tomen conciencia de ello e innoven nuevas técnicas similares y puedan conseguir resultados cada vez mejores para un problema.Por otro lado, no existe un metodo formal de optimizacion de cortes irregulares en la literatura revisada; por este motivo, los metodos

formulados y el algoritmo desarrollado son contribuciones originales en esta linea.

El algoritmo mostrado es de calidad, ya que se ha demostrado con instancias de prueba el error obtenido el cual es 0; con lo cual podemos tratar con este algoritmo instancias de prueba de mayor escala. Pero tenemos que resaltar aca que lo que podria demorar es la simulacion dinamica, ya que para que se compacten las piezas en cada superficie, tiene que pasar un lapso de tiempo, con lo cual el tiempo de procesamiento se incrementaria.

Se pretende que de este trabajo surgan otras tecnicas para mejorar el tiempo de procesamiento de la compactacion de las piezas irregulares como el uso del metodo Bottom Left para el posicionamiento inicial de las piezas, con lo cual el tiempo de compactacion de las piezas en la superficie se reduciria; tambien se podrian realizar trabajos para investigar acerca de otros metodos de deteccion de interferencias, con lo cual se agilizaria el tiempo de procesamiento del algoritmo.

Sería recomendable incentivar más la investigación en esta área, puesto que la mayoría de problemas encontrados en la literatura pueden tener solución utilizando metaheuristicas. Los profesionales de todos los campos del saber podrían contribuir al desarrollo de aplicaciones tecnológicas que resuelvan problemas de diversa complejidad usando Inteligencia Artificial. Asimismo sería necesario que el gobierno y empresas privadas presten apoyo y subvención económica a proyectos realizados en este campo, ya que con su ayuda contribuirían al desarrollo de las investigaciones tecnológicas mejorando así la calidad de los procesos industriales.

Bibliografía

[AL77] A. Albano. A Method to Improve Two-Dimensional Layout. *Computer Aided Design*, 9, 1977.

[APT01] R. Alvarez, A Parajon and J. Tamarit. A Computacional Study of Heuristic Algorithms for Two-Dimensional Cutting Stock Problem. *4th Metaheuristics International Conference*, 2001

[ABJ90] A. Amaral, J. Bernanrdo and J. Jorge. Marker making using automatic placement of irregular shapes for the garment industry. *Computers and Graphics*, 1990, 14(1), 41-46

[AL01] A. Amaral and A. N. Letchford. An improved upper bound for the two-dimensional non-guillotine cutting problem. *Technical Report*, Lancaster University, UK, 2001

[AM95] M. Arenales and R.Morabito. An AND/OR-graph approach to the solution of two-dimensional non-guillotine cutting problems. *European Journal of Operational Research*, 84, 1995, 599-617.

[AS80] A. Alvano and G. Sappupo. Optimal Allocation of Two-Dimensional Irregular Shapes Using Heuristic Search Methods. *IEEE Transactions on Systems, Man and Cybernetics*, 5, 1980, 242-248.

[BCR80] B. Baker, Jr. Coffman and R. Rivest. Orthogonal packings in two dimensions. *SIAM Journal of Computing*, 1980, 9(4), 846-855.

[Be85] J. E. Beasley. An exact two-dimensional non-guillotine cutting tree search procedure. *Operation Research*, v.33, n. 1, 1985, 49-64.

[BDD01] Julia A. Bennell, K. A. Dowsland and W. B. Dowsland. The irregular cutting-stock problem a new procedure for deriving the no-fit polygon. *Computers & Operations Research*, v.28, 2001, 271-287.

[Be82] B. E. Bengtsson. Packing Rectangular Pieces A Heuristic Approach. *The Computer Journal*, v.25, 1982, 353-357.

[BHW93] J. Blazewicz, P. Hawryluk and R. Walkowiak. Using a tabu search approach for solving the two-dimensional irregular cutting problem. *Annals of Operations Research*, v.41, 1993, 313-327.

[BHK+05] E.K. Burke, R. Hellier, G. Kendall and Whitwell. A New Bottom-left-Fill Heuristic Algorithm for the 2D Irregular Packing Problem. *Operations Research*, 2005.

[BMA02] M. A. Boschetti, Mingozzi Aristide and Hadjiconstantinou. New upper bounds for the two-dimensional orthogonal non-guillotine cutting stock problem. *IMA Journal of Management Mathematics*, v.13, 2002, 95-119.

[Br80] D. Brown. An improved BL bound. *Information Processing Letters*, 1980, 11(1), 37-39.

[BB84] M. Biró and E. Boros. Network Flows and Non- Guilhotine Cutting Patterns. *European Journal of Operation Research*, v.16, 1984, 215-221.

bibitem[CGJ82]kn:CGJ82 F. K. R. Chung, M. R. Garey and D. S. Johnson. On packing two-dimensional bins. *SIAM Journal on Algebraic and Discrete Methods*, v.3, n.1, 1982, 66-76.

[Ch83] B. Chazelle. The bottom-left bin-packing heuristic: An efficient implementation. *IEEE Trans. Comput*, v.C-32, n.8, 1983, 697-707.

[DG90] J. J. Daniels and P. Ghandforoush. An Improved Algorithm for the Non-Guillotine-Constrained Cutting-Stock Problem. *Operations Research*, v.41, 1990, 141-149.

[DD92] K. A. Dowsland and W. B. Dowsland. (Packing problems). *European Journal of Operation Research*, v.56, n.1, 1992, 2-14.

[DD93] K. A. Dowsland and W. B. Dowsland. Heuristic approaches to irregular cutting problems. *, Working Paper EBMS/1993/13*, European Business Management School, UC Swansea, UK, 1993.

[DD95] K. A. Dowsland and W. B. Dowsland. Solution approaches to irregular nesting problems. *European Journal of Operation Research*, v.84, 1995, 506-521.

[DD02] K. A. Dowsland and W. B. Dowsland. An Algorithm for polygon placement using botom left strategy. *European Journal of Operation Research*, v.141, 2002, 371-381.

[DB84] D. Dori and B. Bassat. Efficient nesting of congruent convex figures. *Communications of the ACM*, 1984, 228-235.

[DL02] F. Ducatelle and J. Levine. Ant colony optimisation for bin packing and cutting stock problems. http://citeseer.nj.nec.com/506184.html, 2002.

[DKAG 85] H. Dyckhoff, H. J. Kruse, D. Abel and T. Gal. Trim loss and related problems. *Omega*, v.13, 1985, 59-72.

[Dy90] H. Dyckhoff. A typology of cutting and packing problems. *European journal of Operation Research*, v.44, 1990, 145-159.

[DF92] H. Dyckhoff and U. Finke. Cutting and Parking in Production and Distribution. *Physica-Verlag*, 1992.

[DG76] R. G. Dyson and A. S. Gregory. The cutting stock problem in the flat glass industry. *Operational Research Quartely*, v.25, 1976, 41-53.

[Ed71] J. Edmonds. Matroids and the greedy algorithm. *Mathematical Programming* , v.1, 1971, 127-136.

[Fa83] A. A. Farley. Trim loss pattern rearrangement and its relevance to the flat-glass industry. *European jounal of Operation Research*, v.14, 1983, 386-392.

[Fa88] A. A. Farley. Mathematical programming model for cutting stock problems in the clothing industry.. *Journal of the Operation Research Society*, v.39, 1988, 41-53.

[Fa99] L. Faina. An application of simulated anneling to the cutting stock problem. *European jounal of Operation Research*, v.114, 1999, 542-556.

[Fa90] A. A. Farley. The cutting stock problem in the canvas industry. *European Journal of Operation Research*, v.44, 1990, 247-255.

[FC91] L. Figueiredo and P. Carvalho. Introdução à Geometria Computacional. *18º Colóquio Brasileiro de Matemática, Rio de Janeiro*, 1991.

[FS97a] S. P. Fekete and J. Schepers. A New Exact Algorithm for General Orthogonal Ddimensional Knapsack Problems, Algorithms - ESA '97. *Springer Lecture Notes in Computer Science*, V.1284, 1997, 144-156.

[FS97b] S. P. Fekete and J. Schepers. On more-dimensional packing I: Modelling. *Technical Report, Universit¨at zu K¨oln*, 1997, 97-288.

[FR95] Thomas A. Feo and Mauricio G.C. Resende. Greedy Randomized adaptative Search Procedures. *Journal of Global Optimization*, 1995, 1-27.

[FS75] H. Freeman and R. Shapira. Determining the minimun area encasing rectangle for an arbitrary closed curve. *Communications of the ACM*, 1975, 409-413.

[Ga96] V. Garcia. Otimizacao de padroes de corte de chapas de fibra de madeiran reconstituida. *Dissertation*, Deapartamento de Engenharia de Producao, Universidade Federal de Sao Carlos, Brazil, 1996.

[Gi79] P. Gilmore. Cutting stock linear programming, knapsacking, dynamic programming an integer linear programming; some interconnections. *Annals of Discrete Mathematics*, v.4, 1979, 217-235.

[GG65] P. Gilmore and R. Gomory. Multistage cutting stock problems of two and more dimensions. *European Journal of Operation Research*, v.14, 1965, 94-120.

[GG66] P. Gilmore and R. Gomory. The theory and computation of knapsack functions. *European Journal of Operation Research*, v.14, 1966, 1045-1074.

[GO01] A. M. Gomez and J. F. Oliveira. A Grasp Approach to the Nestimg Problem. *MIC'2001 - 4th Metaheuristics International Conference*, 2001.

[GO02] A. M. Gomez and J. F. Oliveira. A 2-exchange heuristic for nesting problems. *European Journal of Operations Research*, v.141, 2002, 359-370.

[HC95] E. Hadjiconstantinou and N. Christofides. An exact algorithm for general, orthogonal, two dimensional knapsack problems. Europ. J. Oper. Research, v. 83, 1995, 39-56.

[HS91] R. Haessler and P. E. Sweeney Cutting stock problems and solution procedures. *European Journal of Operation Research*, v.54, 1991, 141-150.

[HEIS96] I. Harjunkoski, T. Ewsternlund, J. Isaksoon and H.Skrifvars. Different formulations for solving trim-loss problems in a paper converting mill with ILP. *Escape*, v.6, 1996.

[HCK99] P. Healy, M. Creavin and A. Kuusik. An optimal algorithm for rectangle placement. *Operations Research Letters*, v.24, 1999, 73-80.

[HL95] R. Heckmann and T. Lengauer. A simulated annealing approach to the nesting problem in the textile manufacturing industry. *Annals of Operations Research*, v.57, 1995, 103-133.

[Hi80] A. I. Hinxman. The trim-loss and assortment problems: a survey. European journal of Operation Research, v.5 , 1980, 8-18.

[Ho75] J. Holland. Adaptation in Natural and Artificial Systems. *Ann Arbor*, University of Michigan Press, 1975.

[Ho+00] E. Hopper. Two Dimensional Packing utilising evolutionary algorithms and other meta-heuristic methods. *Ph.D. thesis*, University of Wales, Cardiff.

[HT99] E. Hopper and B. Turton. A Genetic Algorithm for a 2D Industrial Packing Problem. *Computers & Industrial Engineering*, v.37, 1999, 375-378.

[IH92] H.S. Ismail and K.K.B. Hon. New approaches for the nesting of two-dimensional shapes for press tool design. *International Journal of Production Research*, v.30,4, 1992, 825-837.

[IS82] S. Israni and J. L. Sanders. Two dimensional cutting stock problem research: A review and new rectangular layout algorithm. *Journal of Manufacturing Systems*, v.1, 1982, 169-182.

[Ja96] S. Jacobs. On Genetic algorithms for the packing of polygons. *European journal of Operation Research*, v.88, 1996, 165-181.

[JFR92] P. Jain, P. Fenyes and R. Richter. Optimal blank nesting using simulated annealing. *Transactions of the ASME*, v.114, 1992, 160-165.

[KP02] J. Kim and F. Pellacini. Jigsaw Image Mosaics. *Computer Graphics*, Cornell University, Siggraph, 2002.

[KGV83] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, v.220, 1983, 671-680.

[KSV91] B. Kröger, P. Schwenderling and O. Vornberger. Parallel genetic packing of rectangles. *In Parallel Problem Solving from Nature 1st Workshop*, Springer Verlag: Berlin, 1991, 160-164.

[Kr95] B. Kröger. Guillotinable bin packing: a genetic approach. *European Journal of Operation Research*, v.84, 1995, 645-661.

[Li77] C.D. Liton A frecuency approach to the one dimensional cutting problem for carpet rolls. Operation Research Quartelly, v.28, 1997, 927-938.

[LT99] D. Liu and H. Teng. An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles. *European journal of Operational Research*, 1999, 112(2) 413-420.

[LM93] Z. Li and V. Milenkovic. A compaction algorithm for non-convex polygons and its application. Proceedings of 9th Annual Symposium on Computation Geometry, ACM, San Diego, CA, 1993, 153-162.

[LMV99] A. Lodi, S. Martello and D. Vigo. Aproximation algorithms for the oriented two-dimensional bin packing problem. *European journal of Operation Research*, v.112, 1999, 158-166.

[LMM02] A. Lodi, S. Martello and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, v.141, 2002, 241-252.

[LMP+92] H. Lutfiyya, B. McMillin, D. A. P. Poshyanon and C.Dagli. Composite stock cutting through simulated annealing. *Mathematical Computer Modelling*, v.16,1, 1992, 57-74.

[Ma79] O.G.B. Madsen. Glass cutting in small firm. *Mathematical Programming*, v. 17, 1979, 85-90.

[MD02] D. Mauricio and R. Delgadillo. Algoritmos FFD y BFD para resolver el problema de cortes de Guillotina. *Technical Report UPG-FISI*, Universidad Nacional Mayor de San Marcos, Nro.01, 2002.

[MM97] D. Mauricio and N. Maculan. A trust region method for zero-onenonlinear programming. *RAIRO Operations Research*, v.31, 1997, 331-341.

[MR02] D. Mauricio and L. Rivera. Una versión eficiente del Algoritmo FFD para resolver el problema de cortes. *UPG - FISI*, Universidad Nacional Mayor de San Marcos (UNMSM); LCMAT -CCT, Universidade Estadual do Norte Fluminense (UENF), 1992.

[Ma03] D. Mauricio. Algoritmos GRASP para el problema de Cortes. *UPG - FISI*, Universidad Nacional Mayor de San Marcos (UNMSM), 2003.

[MDL92] V. Milenkovic, K. Daniels and Z. Li. Placement and compaction of non-convex polygons for clothing manufacture. *Proceedings of the 4th Canadian Conference onComputational Geometry, St. John's, Newfoundland*, 10-14 Agosto, 1992, 236-243.

[MAA92] R. N. Morabito, M. N. Arenales and V. F. Arcaro. An And-Or-Graph approach for two dimensional cutting problems. *European Journal of Operation Research*, v.58, 1992, 263-271.

[Mi88] V. J. Milenkovic. A Positional-Based Physics: Simulating the Motion of Highly Interacting Spheres and Polydra. *Computer Graphics Proceedings-SIGGRAPH88*, 1988, 281-288.

[MG97] R. N. Morabito and V. Garcia. The cutting stock problem in hard: a case study. *Computer Operation Research*, v.25, 1997, 469-485.

[OF90] J. F. Oliveira and J. S, Ferreira. An Improved Version of Wang's Algorithm for Two-Dimensional Cutting Problems. *EJOR*, v44, 1990, 256-266.

[OF93] J. F. Oliveira and J. S, Ferreira. Algorithms for nesting problems. *R.V.V. Vidal (ed.), Applied Simulated Annealing, Lecture Notes in Economics and Mathematical Systems*, v.396, Springer-Verlag, Berlin, 1993, 255-274.

[OF00] J. F. Oliveira and J. S, Ferreira. TOPOS - A new constructive algorithm for nesting problems. *OR Specktrum*, 2000, 22(2), 263-284.

[PMA98] Healy Patrick, Creaving Marcus and kuusik Ago. An Optimal Algorithm for Rectangle Placement. *European journal of Operation Research*, v.24, 1999, 73-80.

[PGD95] V. Parada, A. Gómes and J. De Diego. Exact solutions for constrained two-dimensional cutticng stock problems. *European Journal of Operation Research*, v.84, 1995, 633-644.

[PMG95] V. Parada, R. Muñoz and A. Gómes de Alvarenga. A Hybrid Genetic Algorithm for the Two-Dimensional Guillotine Cutting Problem. Evolutionary Algorithm in Management Applications, Ed. Volker Nissen. Springer Verlag, 1995, ISBN 3-540-60382-4.

[PS85] F. P. Preparata and M. I. Shamos. Computational geometry: an introduction. Springer-Verlag, New York, 1985.

[RA90] D.F. Rogers and J.A. Adams. Mathematical Elements for Computer Graphics. *McGraw-Hill International Editions*, 2do Edition, 1990.

[RST96] J. Riehme, G. Scheithauer and J. Terno. The solution of two-stage guillotine cutting stock problems having extremely varying order demands. *European Journal of Operational Research*, v.91, 1996, 543-552.

[Ri01] L. Rivera. Animación Basada en Física de Modelos Geométricos en Multirresolución. *Tesis de doctorado*, Versión Portugués, Departamento de Informática, Pontificia Universidade Catolica do Rio de Janeiro, Brasil, 2001.

[Ri96] L. Rivera. Simulacion Dinamica de Cuerpos Rigidos con Restricciones de No Interpenetracionde mestrado, Departamento de Inform´atica, Pontif´ýcia Universidade *Tesis de Maestrado*, Catolica (PUC-Rio), Rio de Janeiro, Brasil, 1996.

[RGV01] A. Rocha, R. Gómez and R. Valdivia. Algoritmos para fecho convexo bidimensional. *IMPA - Instituto de matematica pura e aplicada*, 2001.

[RC01] L. Rivera and P. Carvalho. *Animation Based in Dynamic Simulation Involving Irregular Objects with Non-Homogeneous Rugosities*, 2001.

[RCV02] L. Rivera, P. Carvalho and L. Velho. Caixas Orientadas Envolventes na Verificacao de Objetos. *Technical Report TR-02-05*, Nro.02, 2002.

[Sm85] L. Rivera, P. Carvalho and L. Velho. Bin-packing with adaptive search. In: Proceedings of an International Conference on Genetic Algorithms and their Applications. *Grefenstette, ed., Lawrence Erlbaum*, 1985, 202-206.

[SBM+97] R. Sharma, T. Balachander, C. McCord, S. Anand, Q. Zhang. Genetic Algorithms for the Single-Sheet and Multi-Sheet Non-convex Cutting Stock Problem. *Computer Aided Manufacturing Laboratory, Industrial Engineering Program*, University of Cincinnati, 1997.

[Sc99] G. Scheithauer. LP-based bounds for the container and multi-container loading problem. Int. Trans. Oper. Res., v.6, 1999, 199-213.

[TMM88] R.D. Tsai, E.M. Malstrom and H.D. Meeks. A two-dimensional palletizing procedure for warehouse loading operations. *IIE Transactions*, v.20, 1988, 418-425.

[TCL02] F. Tay, T. Chong and F. Lee. Pattern nesting on irregular-shaped stock using Genetic Algorithms. *Engineering Applications of Artificial Intelligence*, v.15, 2002, 551-558.

[VM92] P. Venkateswarlu and C.W. Martyn. The Trim loss problem in a wooden drum industry. *Proceeding of the Convention of Operation Research Society of India*, 1992.

[Vi89] K. V. Viswanathan. New Algorithms for Constrained Rectangular Guillotine Knapsack Problems. *Thesis*, Indian Institute of Management Calcutta, 1989.

[Wa83] P. Y. Wang. Two algorithms for constrained two-dimensional cutting stock problems. *Operations Research*, v.31, 1983, 573-586.

[WIH95] T. Westernlund, J. Isaksoon and Harjunkoski. Solving a production optimization problem in the paper industry. *Report 95-146A, Processs Desing Laboratory, Abo Akademi University*, 1995.

Apéndice A

Segmentos del programa fuente

Aquí presentamos los extractos de codigo fuente citados en el capítulo 8.

```
if((fsys=fopen(arq,"rt"))==NULL)
{
   printf("Erro na apertura de file para leitura");
   return 0;
}

s_objs = aloca_sys_objs();
fgets(token, 40, fsys);
fscanf(fsys,"%s %f", token, &s_objs->t_step);
fscanf(fsys,"%s %u", token, &n);
s_objs->n_frames = n;
fscanf(fsys,"%s %f %f %f", token, &s_objs->grav[0], &s_objs->grav[1], &s_objs->grav[2]);
fscanf(fsys,"%s %f", token, &s_objs->tolerancia_min);
fscanf(fsys,"%s %d", token, &n);

if(s_objs->objs != NULL)  s_objs->objs = libera_objs_todos(s_objs);
if(s_objs->obj_din != NULL)  s_objs->obj_din = libera_obj_din_todos(s_objs);

s_objs->n_objs = n;
s_objs->objs = aloca_objs_todos(s_objs->n_objs);
s_objs->obj_din = aloca_obj_din_todos(s_objs->n_objs);
fscanf(fsys, "%u %s %d", &ob_id, nom_obj, &num_requerimientos);

i = 0;
while(i<s_objs->n_objs && !(feof(fsys)))
{
     ob = &(s_objs->objs[i]);
     ob_din = &(s_objs->obj_din[i]);
     ob->cantidad_objetos = num_requerimientos;
     ob->k = k;
     pathhh = arq+11;
     memset(pathhh, '\0', strlen(arq));
     memcpy(pathhh, nom_obj, strlen(nom_obj));

     fscanf(fsys, "%s %s %d", token, igual, &ob->cor);
     fscanf(fsys, "%s %s %f", token, igual, &w_scale);
     fscanf(fsys, "%s %s %f", token, igual, &ob_din->restitu);

     if(fil_carrega_um_objeto(arq, ob, ob_din, w_scale))
     {
          if (i>0)
              ob->tipoobjeto = 1;
          else
              ob->tipoobjeto = 0;
          fscanf(fsys, "%s %s %f %f %f", token, igual, &ob_din->pos[0], &ob_din->pos[1], &ob_din->pos[2]);
          fscanf(fsys, "%s %s %f", token, igual, &ang);
          ca_calcula_matriz_rotacao(ob_din->rr, ang);
```

Figura A.1: Extracto código funcion filcargadados.

```
    while ((contadorIter <= NumeroIteraciones) && (bandera_igualdad_solucion != 1) )
    {
        sys_contenedores = aloca_sys_contenedores(area_superficie);
        sys_temp = copiaListaPieza(sys_lista_piezas,sys_temp);

        while (DemandasAtendidas(sys_temp)!=0)
        {
            sys_contenedores = grasp_seleccion(sys_contenedores, sys_temp, alfa, beta); //FUNCION DE GRASP CONSTRUCCION
            sys_contenedores = replica_contenedor(sys_contenedores);
            minima_replicacion = get_minima_replicacion(sys_lista_E);
            sys_contenedores->cantidad_superficies = minima_replicacion;
            sys_lista_E = actualizar_demanda(sys_lista_E,minima_replicacion);
            cantidad_contenedores_solucion = cantidad_contenedores_solucion + minima_replicacion;
            cantidad_contenedores_reales++;
            areaocupada = areaocupada + minima_replicacion*sys_contenedores->superficie.area_usada;

            //se agrega otra superficie
            cont_superficie++;
            if (sys_lista_E != NULL) //cuando ya esta vacio no se debe de crear otra superficie
            {
                sys_contenedores = nuevo_nodo_superficie(sys_contenedores, cont_superficie,area_superficie);

            }
            sys_temp = borrar_lista(sys_temp);
            sys_temp = copiaListaPieza(sys_lista_E,sys_temp);

        }

        //verificacion de la condicion de terminacion
        if (c_convergencia <= convergencia)
        {
            if (cantidad_contenedores_reales < comparadorNumContenedores)
            {
                c_convergencia = 1;
                comparadorNumContenedores = cantidad_contenedores_reales;
                *areaocupadatotal = areaocupada;
                sys_contenedoresmejor = borrar_listaContenedores(sys_contenedoresmejor);
                sys_contenedoresmejor = copiaListaContenedores(sys_contenedores,sys_contenedoresmejor);
                *cantidadCont = (int)cantidad_contenedores_reales;

            }
            else c_convergencia++;
        }
        else
        {
            bandera_igualdad_solucion = 1;
        }
```

Figura A.2: Extracto código función graspSeleccion.

```
st_sys_objs *transfcoord_recalcula_coordenadas(st_sys_objs *sys_objs, st_lista *sys_contenedores, st_objetoradio *radios)
{
    st_sys_objs *sys_objs_mod = sys_objs;
    st_lista *l_contenedor = sys_contenedores;
    st_obj_din *obj_din, *obj_din_ant, *obj_din_primer;
    st_objeto *obj;

    float coord_x, coord_y, coord_z;
    int j, cont_objeto_irregular;

    if(sys_objs_mod->objs != NULL)
    {
       cont_objeto_irregular = 0;
       obj_din_primer = &(sys_objs_mod->obj_din[0]);

       for(j=3; j<sys_objs->n_objs; j++) // se debe mostrar por frame el numero de objetos que se posicionaran en cada contenedor

       {
           obj = &(sys_objs_mod->objs[j]);
           obj_din_ant = &(sys_objs_mod->obj_din[j-1]);
           obj_din = &(sys_objs_mod->obj_din[j]);
           // estrategia de posicionamiento vertical de objetos simulando hilo
           if (obj->tipoobjeto == 0)   // es una superficie
           {    // se obtiene las coordenadas del centro del masa del objeto
                coord_x = obj_din_ant->pos[0];
                coord_y = obj_din_ant->pos[1];
                coord_z = obj_din_ant->pos[2];
           }
           if (obj->tipoobjeto == 1)   // es un objeto
           {
                if (cont_objeto_irregular == 0)  //para el primer objeto irregular
                {
                    obj_din->pos[0] = obj_din_primer->pos[0]; //de la superficie primera
                    obj_din->pos[1] = obj_din_ant->pos[1] + radios[j].valor;
                    obj_din->pos[2] = obj_din_ant->pos[2];
                }
                else   //para el resto de objetos segun la formula cj = cj-1 + (r[j-1] + r[j])
                {
                    obj_din->pos[0] = obj_din_primer->pos[0]; //de la superficie primera
                    obj_din->pos[1] = obj_din_ant->pos[1] + radios[j-1].valor + radios[j].valor;
                    obj_din->pos[2] = obj_din_ant->pos[2];
                }
                cont_objeto_irregular ++;
           }


       }
    }

    return sys_objs_mod;
```

Figura A.3: Extracto código función distribuyeobjs.

```
t = t0 + passo_t;
while(step < vezes)
{
   w_ciclo_nao_normal = 0;
   converge = 0;
   tm = t;
   do
   {
      colis = din_dinam_interf_contatos(step, delta_t, n_obs, obs, obs_din, dy, sys_par);
      if(colis < 0)        // interpenetracao ??? ==> ignorar o passo
      {
         din_backup_para_estado(y0);        // recupera o estado anterior
         dd_tt = (c_real32)(dd_tt / 2.0f);  // divide o passo de tempo (passo menor)
         delta_t -= dd_tt;
         w_ciclo_nao_normal = -1;               // indicador de que nao vai ser um ciclo normal
      }
      else
      {
         if(colis > 0)   // em contato (os afastados são ignorados)
         {                    //---->  tratar colisoes e contatos......
             lst = sys_par->lst_par;
             impu_trata_colisoes_main(lst, 1, tm);
             t0 = tm;
             delta_t = t - t0;
             dd_tt = delta_t;
             din_estado_para_backup(y0);

             ttm = fabs(delta_t);
             if(ttm < t_minimo_step)
             {
                 w_ciclo_nao_normal = 0;
                 delta_t = t+ passo_t - t0;
             }
             else
                 w_ciclo_nao_normal = 1;
         }
         else             // nao ha contato,  afastados
         {
             if (w_ciclo_nao_normal)  //  --> estao afastados..avancar um pouco mais...
             {
                 ttm = fabs(t - tm);
                 if(ttm < t_minimo_step)
                 {
                     t0 = tm;
                     w_ciclo_nao_normal = 0;
                     delta_t = t+ passo_t - t0;
                     din_estado_para_backup(y0);
                 }
                 else
                 {
                     w_ciclo_nao_normal = 2;
                     din_backup_para_estado(y0);   // recupera o estado anterior
```

Figura A.4: Extracto código función dinloopmovimiento.

```
PointList * quickHull(PointList *C, PointList *convexHull)
{
  if (C->size > 2)
  {
    Point min_x, max_x;
    PointList C_line, C1, C2, convexHull_1, convexHull_2;
    PointList C1_bkup;
    NodePoint *node;
    pl_create(&C_line);
    pl_create(&C1);
    pl_create(&C2);
    pl_create(&C1_bkup);
    pl_create(&convexHull_1);
    pl_create(&convexHull_2);
    pl_copy(C, &C_line);
    // Obtiene 2 del conjunto. Coordenada Max y Min X
    max_x.x = C_line.first->coord->x;
    max_x.y = C_line.first->coord->y;
    min_x.x = C_line.first->coord->x;
    min_x.y = C_line.first->coord->y;
    for (node = C_line.first; node != NULL; node = node->next)
    {
      if ((node->coord->x < min_x.x) || ((node->coord->x == min_x.x) &&
                                         (node->coord->y < min_x.y)))
      {
        min_x.x = node->coord->x;
        min_x.y = node->coord->y;
      }
      else if ((node->coord->x > max_x.x) || ((node->coord->x == max_x.x) &&
                                              (node->coord->y > max_x.y)))
      {
        max_x.x = node->coord->x;
        max_x.y = node->coord->y;
      }
    }
    // Obtiene un conjunto de puntos a la izquierda de (min_x, max_x)
    getPointsAtLeft(&C_line, &min_x, &max_x, &C1);
    pl_copy(&C1, &C1_bkup);
    // // Obtiene un conjunto de puntos a la izquierda de (max_x, min_x)
    getPointsAtLeft(&C_line, &max_x, &min_x, &C2);
    qH(&C1, &convexHull_1, &min_x, &max_x);
    qH(&C2, &convexHull_2, &max_x, &min_x);
    concatConvexHull(&convexHull_1, &convexHull_2, convexHull, &min_x, &max_x);
    pl_destroy(&convexHull_2);
    pl_destroy(&convexHull_1);
    pl_destroy(&C1_bkup);
    pl_destroy(&C2);
    pl_destroy(&C1);
    pl_destroy(&C_line);
  }
  else
  {
```

Figura A.5: Extracto código función quickHull.

Apéndice B

Resultados de instancias de prueba

Aqui presentamos las instancias de prueba del capítulo 9

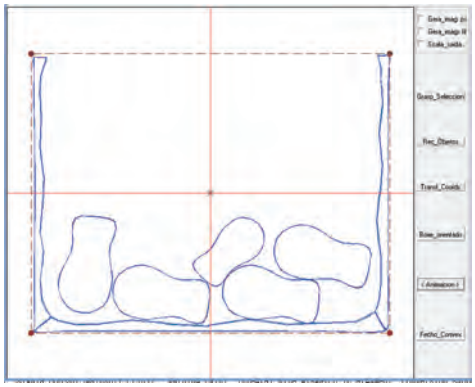
A continuación se muestra el resultado de las instancias de prueba del capítulo 9.

Instancias de Prueba		
Nombre	**Objetos**	**Cantidad**
Instancia 1	lingua32	2
	let1	1
	lingua232	2
	lingua32	4
	let1	2
Instancia 2	ligua32	2
	let1	4
	lingua232	3
	lingua32	1
	let1	1
Instancia 3	ligua32	3
	let1	2
	lingua232	3
	lingua32	2
	let1	4
Instancia 4	ligua32	1
	let1	2
	lingua232	3
	lingua32	4
	let1	4
Instancia 5	ligua32	2
	let1	1
	lingua232	2
	lingua32	4
	let1	5

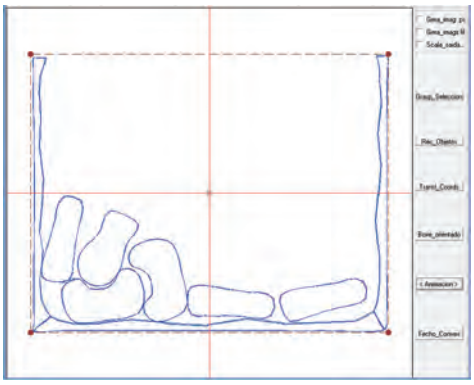
Cuadro B.1: Tabla de Instancias de Prueba 1 al 5

Instancias de Prueba		
Nombre	**Objetos**	**Cantidad**
Instancia 6	lingua32	3
	let1	1
	lingua232	3
	lingua32	2
	let1	5
	lingua232	1
Instancia 7	lingua32	4
	let1	1
	lingua232	1
	lingua32	2
	let1	4
	lingua232	2
Instancia 8	lingua32	3
	let1	2
	lingua232	1
	lingua32	2
	let1	2
	lingua232	1
Instancia 9	lingua32	3
	let1	1
	lingua232	1
	lingua32	1
	let1	2
	lingua232	2
Instancia 10	lingua32	2
	let1	2
	lingua232	4
	lingua32	2
	let1	4
	lingua232	2

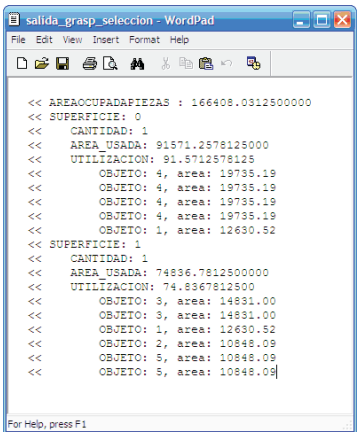
Cuadro B.2: Tabla de Instancias de Prueba 6 al 10



Superficie 1

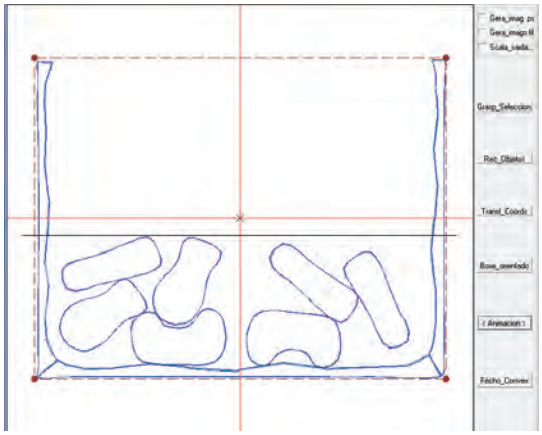


Superficie 2

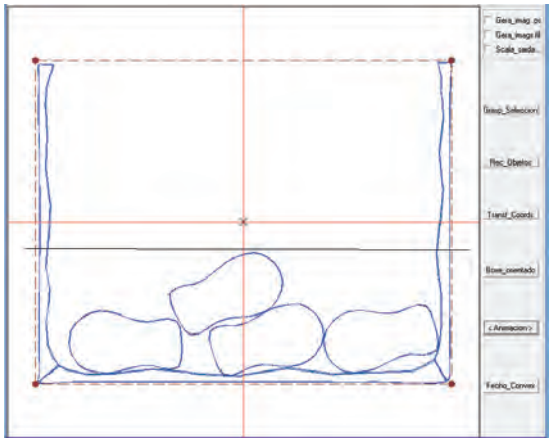


Resultado Algoritmo

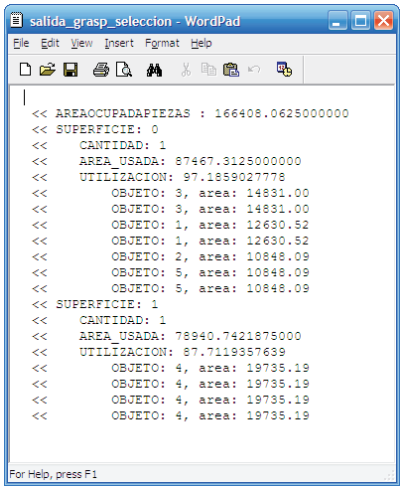
Figura B.1: Figura Resultado Ejecución Instancia 1



Superficie 1

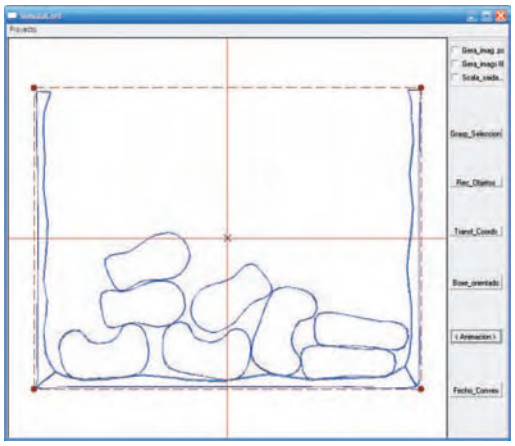


Superficie 2

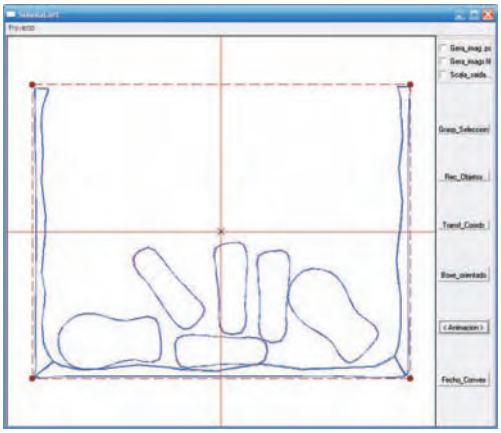


Resultado Algoritmo

Figura B.2: Figura Resultado Ejecución Instancia 2



Superficie 1



Superficie 2

salida_grasp_seleccion - WordPad

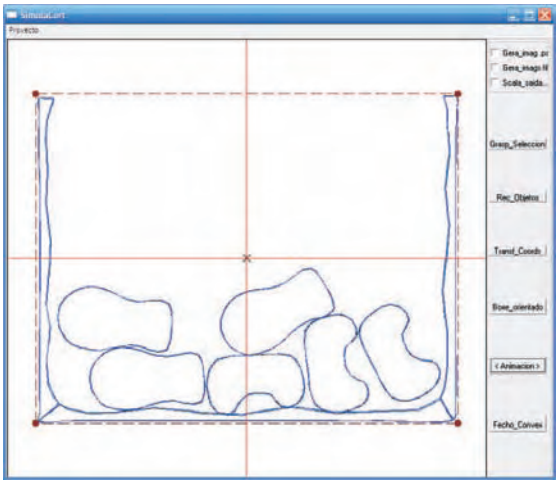
File Edit View Insert Format Help

```
<< AREAOCUPADAPIEZAS : 186943.4687500000
<< SUPERFICIE: 0
<<    CANTIDAD: 1
<<    AREA_USADA: 104080.7343750000
<<    UTILIZACION: 99.1245089286
<<        OBJETO: 3, area: 14831.00
<<        OBJETO: 3, area: 14831.00
<<        OBJETO: 3, area: 14831.00
<<        OBJETO: 1, area: 12630.52
<<        OBJETO: 1, area: 12630.52
<<        OBJETO: 1, area: 12630.52
<<        OBJETO: 2, area: 10848.09
<<        OBJETO: 2, area: 10848.09
<< SUPERFICIE: 1
<<    CANTIDAD: 1
<<    AREA_USADA: 82862.7187500000
<<    UTILIZACION: 78.9168750000
<<        OBJETO: 5, area: 10848.09
<<        OBJETO: 4, area: 19735.19
<<        OBJETO: 4, area: 19735.19
<<        OBJETO: 5, area: 10848.09
<<        OBJETO: 5, area: 10848.09
<<        OBJETO: 5, area: 10848.09
```

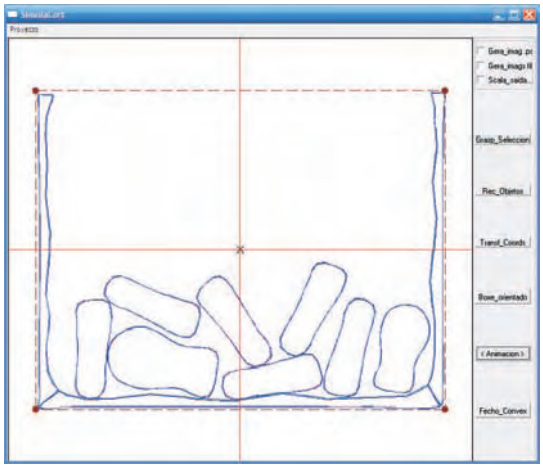
For Help, press F1

Resultado Algoritmo

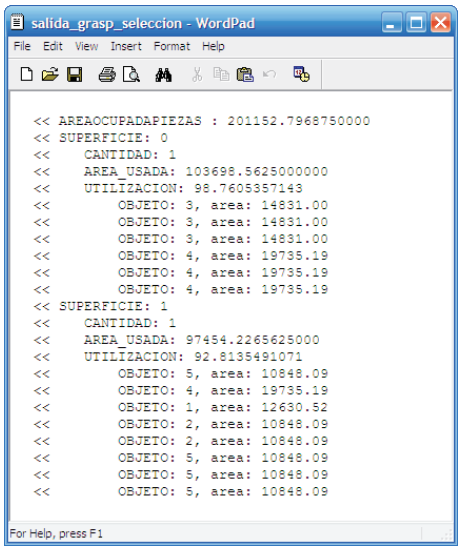
Figura B.3: Figura Resultado Ejecución Instancia 3



Superficie 1

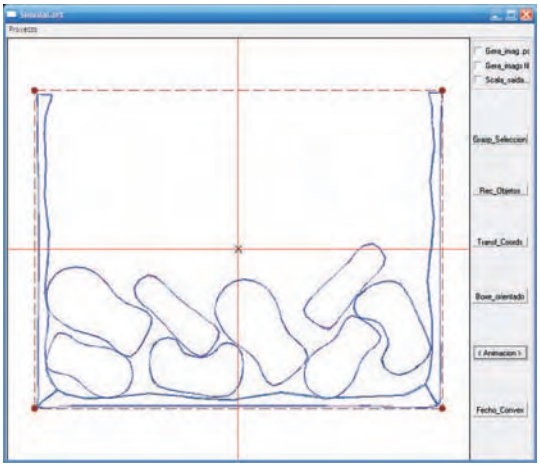


Superficie 2

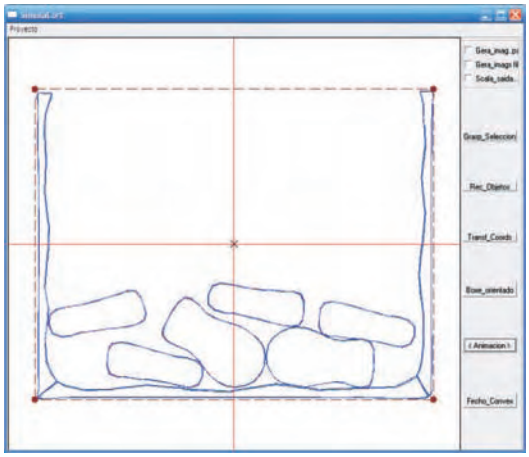


Resultado Algoritmo

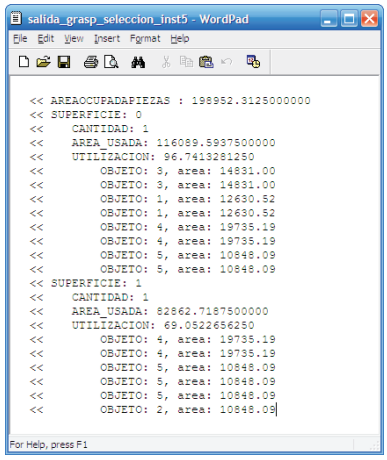
Figura B.4: Figura Resultado Ejecución Instancia 4



Superficie 1

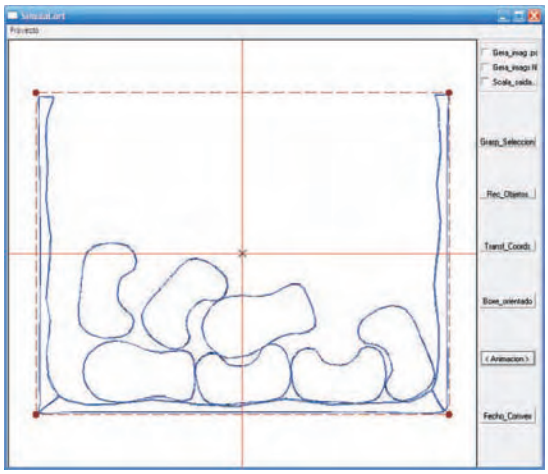


Superficie 2

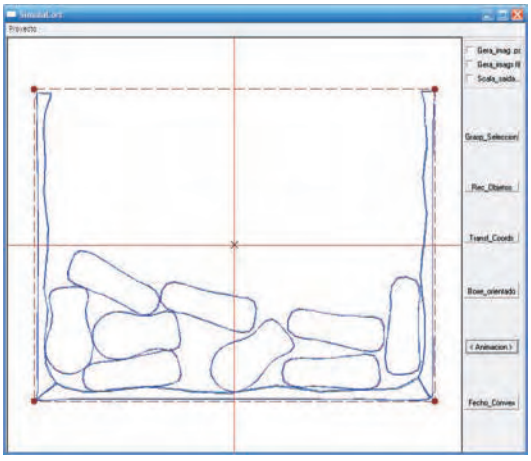


Resultado Algoritmo

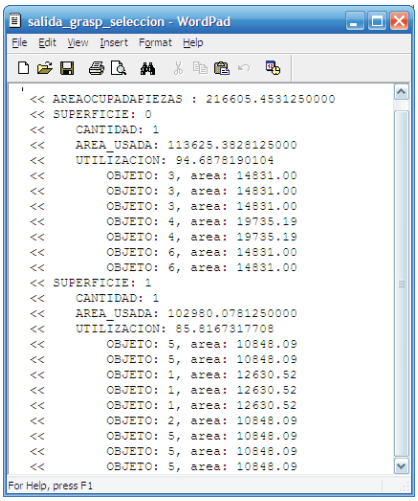
Figura B.5: Figura Resultado Ejecución Instancia 5



Superficie 1

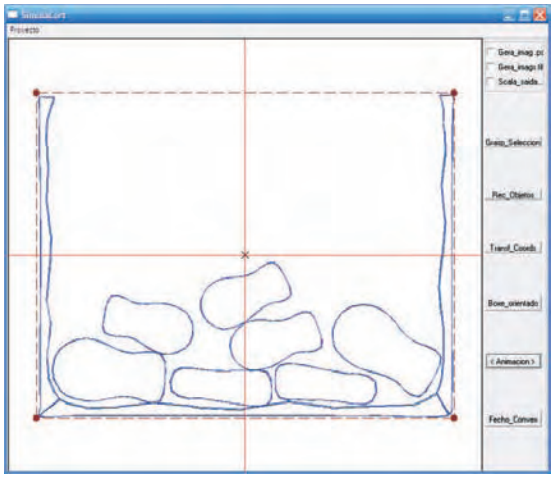


Superficie 2

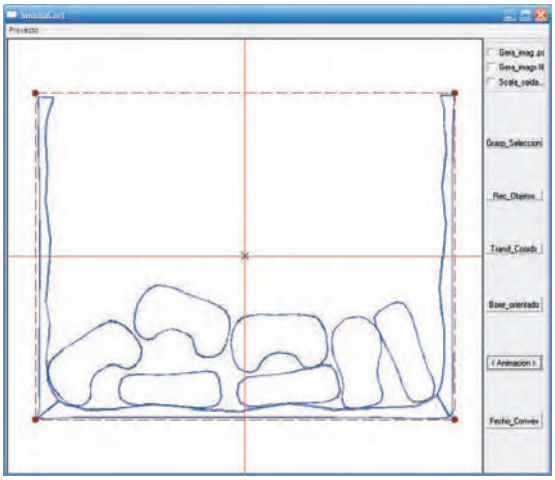


Resultado Algoritmo

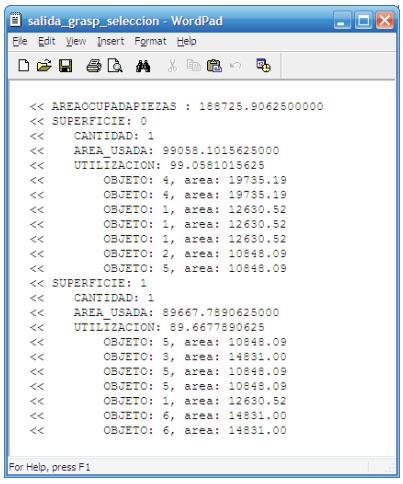
Figura B.6: Figura Resultado Ejecución Instancia 6



Superficie 1

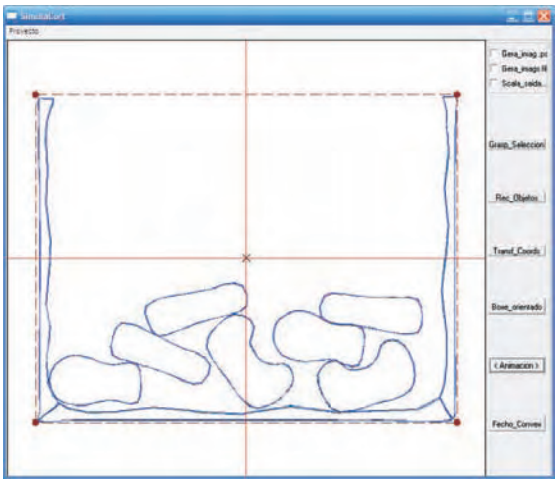


Superficie 2

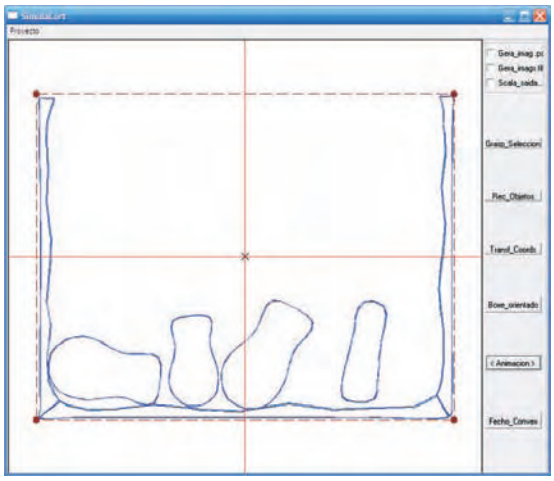


Resultado Algoritmo

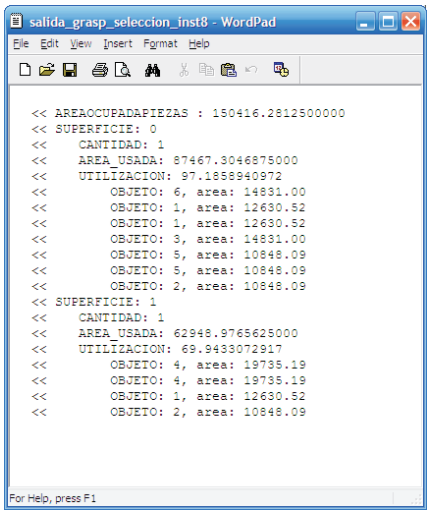
Figura B.7: Figura Resultado Ejecución Instancia 7



Superficie 1

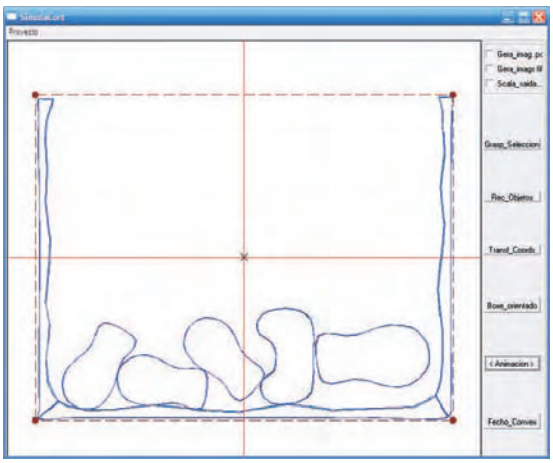


Superficie 2

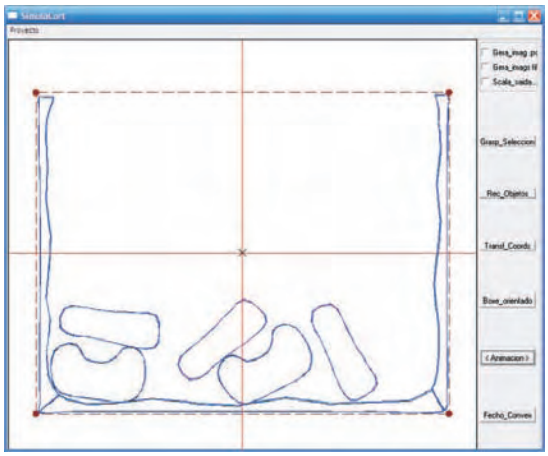


Resultado Algoritmo

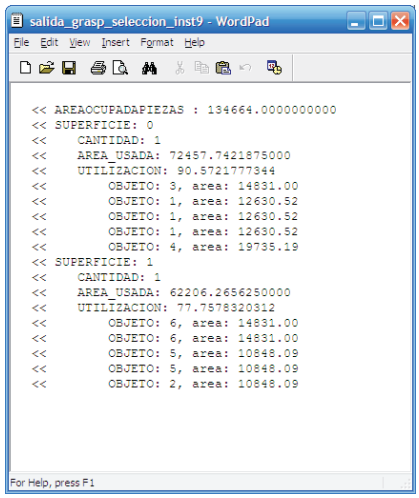
Figura B.8: Figura Resultado Ejecución Instancia 8



Superficie 1

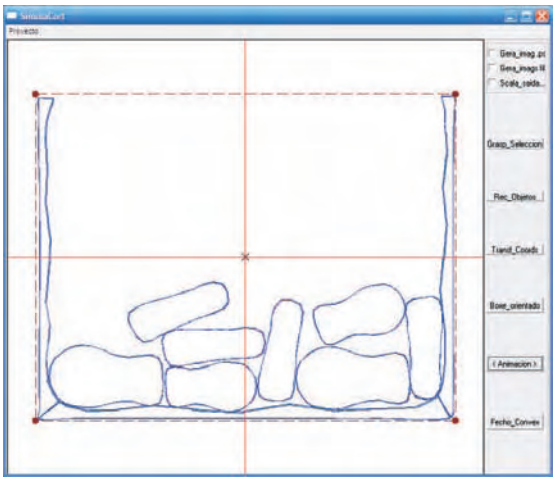


Superficie 2

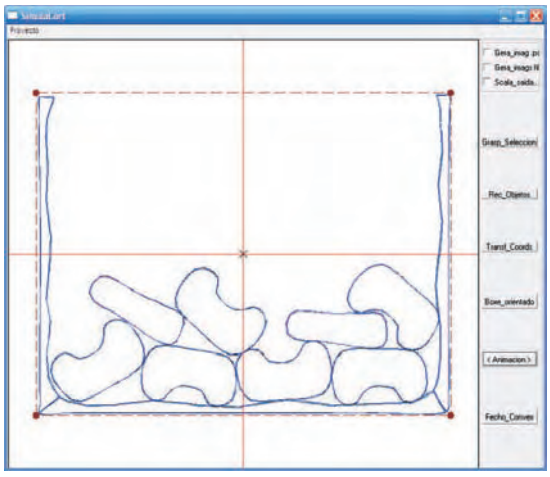


Resultado Algoritmo

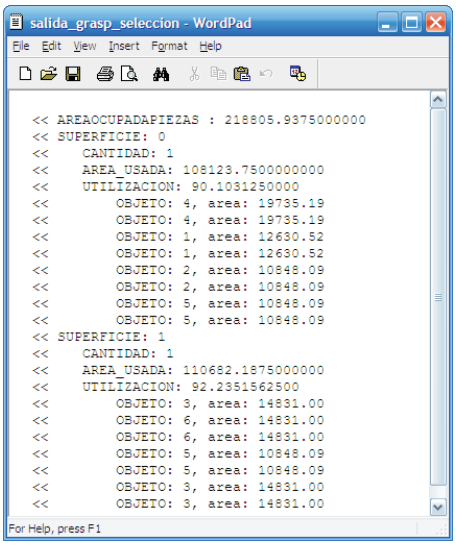
Figura B.9: Figura Resultado Ejecución Instancia 9



Superficie 1



Superficie 2



Resultado Algoritmo

Figura B.10: Figura Resultado Ejecución Instancia 10